VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Measurement and Control

Artificial intelligence in mobile autonomous sensor networks

Doctoral thesis

Ostrava, 2010 Ing. Andrej BENCÚR

Hereby I wish to thank my wife for all the support during my work on this thesis.
I would like to thank Prof.Dr.Ing. Miroslav Pokorný and Dr.Jan Smid for their advice and guidance.

ABSTRACT

This work is concerned with a common and frequent engineering task of constructing a least-squares model based on measured data. What is provided are data from one or more sensors. Also is known that an unknown model is generated by set of functions that come from a known family of function. However it is unknown ahead of time what subset of functions and what concrete parameters generated the data. The task is to propose a method of obtaining this model using minimal number of measurements. This method also provides rules for determination of numerical values of variance and error that let us make a decision whether the number of measurements is sufficient, or additional measurements need to be done, thus the variance of the model is used as the stopping criterion in this case.

The proposed analysis is relevant in practical situations where consistency theorems do not give necessary guidance for small samples of data.

The method of least-squares provides a model that can be accurate (small total error). This model is optimal in the least-squares sense, but the variance of the model is not guaranteed and can be large in certain sub-domain. In case of large data samples consistency theorems guarantee convergence to the true model. However, small data samples can result in a catastrophic model. To avoid this kind of models additional measurements are required. The purpose of these measurements is to decrease the model variance that is indicative of a misleading model. Use of a minimization procedure that tests potential future measurement points and selects an optimal or sub-optimal data point with a good impact on the model variance is proposed. The non-linear parameters of the model are calculated by genetic algorithm.

Part of this work is also an algorithm for searching the maximum admissible variance for a group of models and implementation of algorithms in MATLAB and C#. For practical verification wireless control of an autonomous robot based on image recognition was elaborated.

ABSTRAKT

Tato práce se zabývá častým a běžným inženýrským úkolem sestavení modelu pomocí metody nejmenších čtverců na základě měřených dat. K dispozici jsou data z jednoho nebo více senzorů. Taktéž se ví, že neznámý model je generován sadou funkcí, které vzešly ze známé skupiny funkcí. Přesto není předem známa podmnožina funkcí a konkrétní parametry, které generovaly data. Cílem práce je návrh metody identifikace tohoto modelu pomocí minimálního počtu měření. Tato metoda také poskytne pravidla pro určení číselných hodnot variance a chyby, které umožní učinit rozhodnutí, zda je současný počet měřicích bodů dostatečný, nebo zda je potřeba přidat další měřicí body, tudíž variance modelu v tomto případě slouží zároveň jako kritérium ukončení.

Navržený algoritmus je vhodný pro situace, v nichž teorémy o konzistenci neposkytují potřebný návod pro tvorbu modelů pomocí malého množství dat.

Metoda nejmenších čtverců poskytne přesný model s malou celkovou chybou. Tento model je optimální ve smyslu nejmenších čtverců, ale variance modelu může být v některé oblasti modelu velmi velká. V případě velkého množství dat teorémy o konzistenci zaručují konvergenci hledaného modelu ke skutečnému modelu. Model sestavený z malého počtu vzorků může být ale ve výsledku velmi nepřesný přes malou chybu. Aby bylo možné vyhnout se této situaci, je nutné přidat další měřicí bod. Cílem přidání měřicího bodu je zmenšení variance, která vypovídá o možné nesprávnosti modelu. Pomocí navržené minimalizační procedury, která testuje potenciální nové měřicí body je zvolen optimální, nebo sub-optimální měřicí bod s dobrým vlivem na varianci modelu. Tato metoda byla nazvána prediktivní minimalizací variance. Nelineární parametry modelu jsou počítány pomocí genetického algoritmu.

Součástí práce je také algoritmus hledání maximální povolené variance pro skupinu modelů a implementace algoritmů v Matlabu a v jazyce C#. Pro praktické ověření bylo rozpracováno bezdrátové řízení autonomního robota na základě rozpoznávání obrazu.

LIST OF SYMBOLS

w	weight of polynomial or radial basis function
$\widehat{w}$	estimate of weight
c	center of radial basis function
s	width of radial basis function
y	mathematical model
θ	parameter of model
φ, ϕ	basis function
i	index variable
e	residuals of model
ϵ	independent random variable
n	number of data points
m	number of basis functions
y_p	predicted value
E	expectation
GA	genetic algorithm
A	original location of the robot
B	orientation location of the robot
C	target location of the robot
α	robot rotation angle
k	scaling between the image screen and physical space
$\boldsymbol{u}, \boldsymbol{v}$	vectors used for robot's navigation

LIST OF SYMBOLS

Contents

1 INTRODUCTION

Constructing models from observed data is a fundamental element in science. Several methodologies and nomenclatures have been developed in different application areas. In the control area, the techniques are known under the term System Identification.

The main goal of this thesis is to design a method of measurements and finding of appropriate model of a static physical field based on small number of samples which can be later extended and used in a sensor network.

The novel approach and the main contribution of this work lie in the use of variance and predictive variance method for finding accurate model with a small number of measurements and use of genetic algorithms for optimization of the parameters of the model's basis functions.

Building Models

Basically, a model has to be constructed from observed data [1]. The mental model of car-steering dynamics, for example, is developed through driving experience. Graphical models are made up from certain measurements. Mathematical models may be developed along two routes (or a combination of them). One route is to split up the system, figuratively speaking, into subsystems, whose properties are well understood from previous experience. This basically means that we rely on "laws of nature" and other well-established relationships that have their roots in earlier empirical work. These subsystems are then joined mathematically and a model of the whole system is obtained. This route is known as *modeling* and does not necessarily involve any experimentation on the actual systems. The procedure of modeling is quite application dependent and often has its roots in tradition and specific techniques in the application area in question. Basic techniques typically involve structuring of the process into block diagrams with blocks consisting of simple elements. The reconstruction of the system from these simple blocks is now increasingly being done by computer, resulting in a software model rather than a mathematical model.

The Fiction of a True System

The real-life actual system is an object of a different kind than our mathematical models. In a sense, there is an impenetrable but transparent screen between our world of mathematical descriptions and the real world. We can look through a window and compare certain aspects of the physical system with its mathematical description, but we can never establish any exact connection between them. The question of nature's susceptibility to mathematical description has some deep philosophical aspects, and in practical terms we have to take a more pragmatic view of models. Our acceptance of models should thus be guided by "'usefulness" rather than "truth." Nevertheless, we shall occasionally use a concept of "the true system," defined in terms of a mathematical description. Such a fiction is helpful for devising identification methods and understanding their properties. In such contexts we assume that the observed data have been generated according to some well-defined mathematical rules, which of course is an idealization.

Three Basic Entities

The construction of a model from data involves three basic entities:

1) The data
2) A set of candidate models
3) A rule by which candidate models can be assessed using the data

Let us comment on each of these:

1) The data record. The input-output data are sometimes recorded during a specifically designed identification experiment, where the user may determine which signals to measure and when to measure them and may also choose the input signals. The object with experiment design is thus to make these choices so that the data become maximally informative, subject to constraints that may be at hand. In other cases the user may not have the possibility to affect the experiment, but must use data from the normal operation of the system.

2) The set of models. A set of candidate models is obtained by specifying within which collection of models we are going to look for a suitable one. This is no doubt the most important and at the same time, the most difficult choice of the identification of a system procedure. It is here that a priori knowledge and engineering intuition and insight have to be combined with formal properties of models. Sometimes the model set is obtained after careful modeling. Then a model with some unknown physical parameters is constructed from basic physical laws and other well-established relationships. In other cases standard linear models may be employed, without reference to the physical background. Such a model set, whose parameters are basically viewed as vehicles for adjusting the fit to the data and do not reflect physical considerations in the system, is called a black box. Model sets with adjustable parameters with physical interpretation may, accordingly, be called gray boxes.

3) Determining the "best" model in the set, guided by the data. This is the identification method. The assessment of model quality is typically based on how the models perform when they attempt to reproduce the measured data.

Model Validation

After having settled on the preceding three choices, we have, at least implicitly, arrived at a particular model: the one in the set that best describes the data according to the chosen criterion. It then remains to test whether this model is "good enough," that is whether it is valid for its purpose. Such tests are known as *model validation.* They involve various procedures to assess how the model relates to observed data, to prior knowledge, and to its intended use. Deficient model behavior in these respects makes us reject the model, while good performance will develop a certain confidence in the model. A model can never be accepted as a final and true description of the system. Rather, it can at best be regarded as a good enough description of certain aspects that are of particular interest to us.

The System Identification Loop

The system identification procedure has a natural logical flow: first collect data, then choose a model set, then pick the "best" model in this set. It is quite likely, though, that the model first obtained will not pass the model validation tests. We must then go back and revise the various steps of the procedure. The model may be deficient for a variety of reasons:

- The numerical procedure failed to find the best model according to our criterion.
- The criterion was not well chosen.
- The model set was not appropriate, in that it did not contain any "good enough" description of the system.
- The data set was not informative enough to provide guidance in selecting good models.

The major part of an identification application in fact consists of addressing these problems, in particular the third one, in an iterative manner, guided by prior information and the outcomes of previous attempts. See Figure 1.1. Interactive software obviously is an important tool for handling the iterative character of this problem.

To make use of the loop in the Fig. 1.1, the user has to be familiar with a number of things:

1) Available techniques of identification and their rationale, as well as typical choices of model sets
2) The properties of the identified model and their dependence on the basic items: data, model set and identification criterion
3) Numerical schemes for computing the estimate
4) How to make intelligent choices of experiment design, model set, and identification criterion, guided by prior information as well as by observed data

In fact, a user of system identification may find that he or she is primarily a user of an interactive identification software package. Items 1) and 3) are then part of the package and

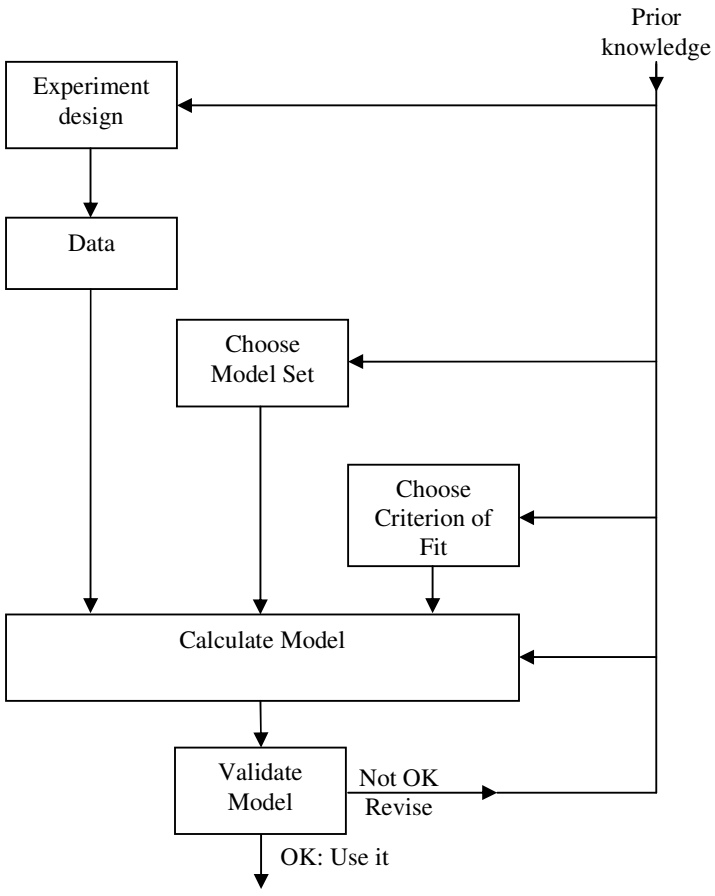


Figure 1.1 The system identification loop

the important thing is to have a good understanding of item 2) so that task 4) can be successfully completed.

The core of identification of a system consists of relatively few fundamental results of statistical nature around the concepts of *information, estimation (learning)* and *validation (generalization).* Like planets in the solar system, the satellites offer different reflections of the radiation from the core.

1.1 The core of estimating models

The core of estimating models is statistical theory [2]. It evolves around the following concepts:

Model

This is a relationship between observed quantities. In loose terms, a model allows for prediction of properties or behaviors of the object. Typically the relationship is a mathematical expression, but it could also be a table or a graph. Model may be denoted by m.

True Description

Even though in most cases it is not realistic to achieve a "true" description of the object to be modeled, it is sometimes convenient to assume such a description as an abstraction. It is of the same character as a model, but typically much more complex. It can be denoted by S.

Model Class

This is a set, or collection, of models. It will generically be denoted by $\mathcal{M}$. It could be a set that can be parameterized by a finite-dimensional parameter, like "all linear state-space models of order *n*", but it does not have to, like "all surfaces that are piecewise continuous".

Complexity

This is a measure of "size" or "flexibility" of a model class. We shall use the symbol C for complexity measures. This could be the dimension of a vector that parameterizes the set in a smooth way, but it could also be something like "the maximum norm of the Hessian of all surfaces in the set."

Information

This concerns both information provided by the observed data and prior information about the object to be modeled, like a model class.

Estimation

This is the process of selecting a model guided by the information. The data used for selecting the model is called *Estimation Data,* (or *training data)* and will be denoted by Z_e^N (with *N* marking the size of the data set). It has become more and more fashionable to call this process *learning,* also among statisticians.

Validation

This is the process of ensuring that the model is useful not only for the estimation data, but also for other data sets of interest. Data sets for this purpose are called *validation data,* to be denoted by Z_v. Another term for this process is *generalization.*

Model Fit

This is a (scalar) measure of how well a particular model *m* is able to "explain" or "fit to" a particular data set *Z*. It will be denoted by $\mathcal{F}$ (m, Z).

To have a concrete picture of a template estimation problem, it could be useful to think of elementary *curve – fitting*.

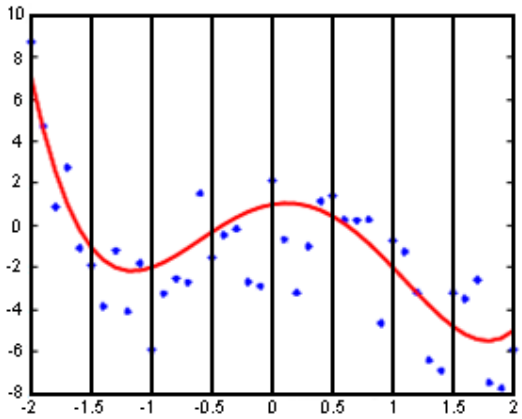


Figure 1.2 A Template Problem - Curve Fitting

Consider an unknown function $g_0(x)$. For a sequence of x-values (regressors) $\{x_1, x_2, \dots, x_N\}$ (that may or may not be chosen by the user) we observe the corresponding function values with some noise:

$$y(t) = g_0(x_t) + e(t) \tag{1.1}$$

The problem is to construct an estimate

$$\hat{g}_N(x) \tag{1.2}$$

from

$$Z^N = \{y(1), x_1, y(2), x_2, \dots, y(N), x_N,\} \tag{1.3}$$

This is a well known basic problem encountered often. In most applications, *x* is a vector of dimension, say, *n*. This means that *g* defines a surface in $\mathbb{R}^{n+1}$if *y* is scalar. If *y(k)* itself is a *p*-dimensional vector, it is in this perspective convenient to view the problem as *p* separate surface-fitting problems, one for each component of *y*.

Two typical approaches are the following ones:

Parametric: Postulate a parameterized model set $\mathcal{M}$, of say *d-1th* order polynomials *g(x, θ)*, parameterized by the *d* coefficients θ (for a scalar *x*), and then adjust θ to minimize the least squares fit between *y(k)* and $g(x_t, \theta)$. A complexity measure *C* could be the order *n*.

Nonparametric: Form, at each x, a weighted average of the neighboring $y(k)$. Then a complexity measure C could be the size of the neighborhoods. (The smaller the neighborhoods, the more complex/flexible curve.)

The border line between these approaches is not necessarily distinct.

2 ESTIMATION

All data sets contain both useful and irrelevant information ("Signal and noise") [2]. In order not to get fooled by the irrelevant information it is necessary to meet the data with a prejudice of some sort. A typical prejudice is of the form "Nature is Simple". The conceptual process for estimation then becomes

$$\hat{m} = \arg\min_{\mathrm{m}\in\mathcal{M}}[\mathcal{F}(m, Z_e^N) + h(C(m), N)] \tag{2.1}$$

where $\mathcal{F}$ is the chosen measure of fit, and *h(C(m),N)* is a penalty based on the complexity of the model m or the corresponding model set $\mathcal{M}$ and the number of data. That is, the model is formed taking two aspects into account:

1) The model should show good agreement with the estimation data
2) The model should not be too complex

These aspects are somewhat contradictory, and a good trade-off must be found, as we shall discuss later. Since the "information" (at least the irrelevant part of it) typically is described by random variables, the model *m* will also become a random variable.

The method (2.1) has the flavor of a parametric fit to data. However, with a conceptual interpretation it can also describe non-parametric modeling, like when a model is formed by kernel smoothing of the observed data.

The complexity penalty could simply be that the search for a model is constrained to model sets of adequate simplicity, but it could also be more explicit as in the curve-fitting problem:

$$V_N(\theta, Z_e^N) = \sum\big(y(t) - g(\theta, x_t)\big)^2 \tag{2.2}$$

$$\hat{\theta}_N = \arg\min_\theta V_N(\theta, Z_e^N) + \delta \parallel \theta \parallel^2 \tag{2.3}$$

Such model complexity penalty terms as in (2.3) are known as *regularization* terms.

2.1 Fit to validation data

It is not too difficult to find a model that describes estimation data well [2]. With a flexible model structure, it is always possible to find something that is well adjusted to data. The real test is when the estimated model is confronted with a new set of data - validation data. The average fit to validation will be worse than the fit to estimation data. There are several analytical results that quantify this deterioration of fit. They all have the following conceptual form: Let a model $\hat{m}$ be estimated from an estimation data set Z_e^N in a model set $\mathcal{M}$. Then

$$\mathcal{F}(\hat{m}, Z_v) = \mathcal{F}(\hat{m}, Z_e^N) + f(C(\mathcal{M}), \mathcal{N}) \tag{2.4}$$

Here, the left hand side denotes the expected fit to validation data, while the first term on the right is the model's actual fit to estimation data ("the empirical risk"). The fit is

typically measured as the mean square error as in (2.2). The quantity *f* is a strictly positive function which increases with the complexity *C* and decreases with the number *N* of estimation data. Hence, to assess the quality of the model one has to adjust the fit seen on the estimation data with this positive quantity. The more flexible the model set, the more deterioration of the fit should be expected. Note that $\widehat{m}$ is a random variable, so the statement (2.4) is a probabilistic one.

2.2 Polynomial curve fitting

Many of the important issues concerning the modeling can be introduced in the simpler context of polynomial curve fitting [3]. Here the problem is to fit a polynomial to a set of *N* data points by the technique of minimizing an error function. Consider the M^{th}-order polynomial given by

$$y(x) = w_0 + w_1 x + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{2.5}$$

This can be regarded as a non-linear mapping which takes *x* as input and produces *y* as output. The precise form of the function *y(x)* is determined by the values of the parameters $w_0 \ldots w_M$. It is convenient to denote the set of parameters$(w_0 \ldots w_M)$ by the vector $\boldsymbol{w}$. The polynomial can then be written as a functional mapping in the form

$$y = y(x; \boldsymbol{w}) \tag{2.6}$$

We shall label the data with the index $n = 1, \ldots, N$ so that each data point consists of a value of *x,* denoted by x^n, and a corresponding desired value for the output *y,* which we shall denote by t^n. In order to find suitable values for the coefficients in the polynomial, it is convenient to consider the error between the desired output t^n, for a particular input x^n, and the corresponding value predicted by the polynomial function given by $y(x^n, \boldsymbol{w})$. Standard curve-fitting procedures involve minimizing the square of this error, summed over all data points, given by

$$E = \frac{1}{2}\sum_{n=1}^{N}\{y(x^n; \boldsymbol{w}) - t^n\}^2 \tag{2.7}$$

We can regard *E* as being a function of $\boldsymbol{w}$, and so the polynomial can be fitted to the data by choosing a value for $\boldsymbol{w}$, which we denote by $\widehat{\boldsymbol{w}}$, which minimizes *E.* Note that the polynomial (2.5) is a linear function of the parameters $\boldsymbol{w}$ and so (2.7) is a quadratic function of $\boldsymbol{w}$. This means that the minimum of *E* can be found in terms of the solution of a set of linear algebraic equations. Functions which depend linearly on the adaptive parameters are called linear models, even though they may be non-linear functions of the original input variables.

We can illustrate the technique of polynomial curve fitting by generating synthetic data in a way which is intended to capture some of the basic properties of real data sets used in pattern recognition problems. Specifically, we generate sample data from the function

$$h(x) = 0.5 + 0.4\sin(2\pi x) \tag{2.8}$$

by sampling the function $h(x)$ at equal intervals of x and then adding random noise with a Gaussian distribution having standard deviation $a = 0.05$. Thus for each data point a new value for the noise contribution is chosen. A basic property of most data sets of interest in pattern recognition is that the data exhibits an underlying systematic aspect, represented in this case by the function $h(x)$ but is corrupted with random noise. The central goal in model estimation is to produce a system which makes good predictions for new data, in other words one which exhibits good generalization. In order to measure the generalization capabilities of the polynomial, we have generated a second data set called a *test set*, which is produced in the same way as the training set, but with new values for the noise component. This reflects the basic assumption that the data on which we wish to use the pattern recognition system is produced by the same underlying mechanism as the training data. The best generalization to new data is obtained when the mapping represents the underlying systematic aspects of the data, rather capturing the specific details (i.e. the. noise contribution) of the particular training set. We will therefore be interested in seeing how close the polynomial $y(x)$ is to the function $h(x)$.

Figure 2.1 shows the 11 points from the training set, as well as the function

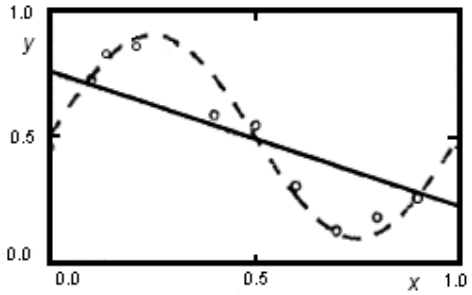


Figure 2.1. An example of a set of 11 data points obtained by sampling the function $h(x)$

$h(x)$ from (2.8), at equal intervals of x and adding random noise. The dashed curve shows the function $h(x)$, while the solid curve shows the rather poor approximation obtained with a linear polynomial, corresponding to $M = 1$ in (2.5).

As can be seen, this polynomial gives a poor representation of $h(x)$, as a consequence of its limited flexibility. We can obtain a better fit by increasing the order of the polynomial, since this increases the number of *degrees of freedom* (i.e. the number of free parameters) in the function, which gives it greater flexibility.

Figure 2.2 shows the result of fitting a cubic polynomial $(M = 3)$ which gives a much better approximation to $h(x)$. If, however, we increase the order of the polynomial too far, then the approximation to the underlying function actually gets worse.

Figure 2.3 shows the result of fitting a 10th-order polynomial $(M = 10)$. This is now able to achieve a perfect fit to the training data, since a 10th-order polynomial has 11 free parameters, and there are 11 data points. However, the polynomial has fitted the data by developing some dramatic oscillations. Such functions are said to be *over-fitted* to the data. As a consequence, this function gives a poor representation of $h(x)$.

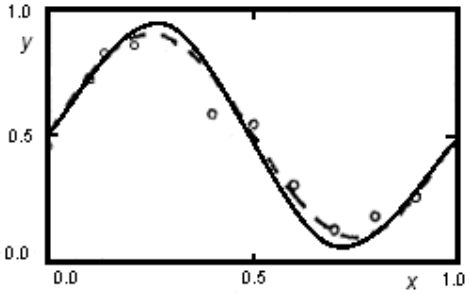


Figure 2.2 Same data set as in Fig. 2.1, but fitted by a cubic (M = 3) polynomial

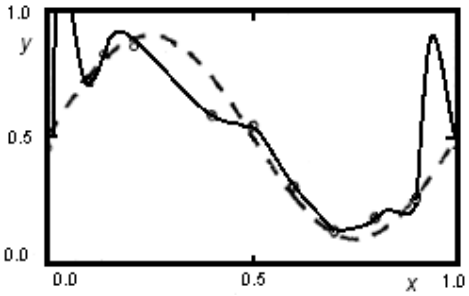


Figure 2.3 The result of fitting the same data set as in Fig. 2.1 using a 10th- order (M = 10) polynomial

Fig. 2.4 shows a plot of E^{RMS} (Root mean square error) for both the training data set and the test data set, as a function of the order M of the polynomial. We see that the training set error decreases steadily as the order of the polynomial increases. The test set error, however, reaches a minimum at $M = 3$, and thereafter increases as the order of the polynomial is increased.

The ability of the polynomial to generalize to new data (i.e. to the test set) therefore reaches an optimum value for a polynomial of a particular degree of complexity. A model which has too little flexibility, such as the linear polynomial of Fig.2.1, has a high bias, while a model which has too much flexibility, such as the l0th-order polynomial of Fig.2.3, has a high variance. The point of best generalization is determined by the trade-off between these two competing properties, and occurs

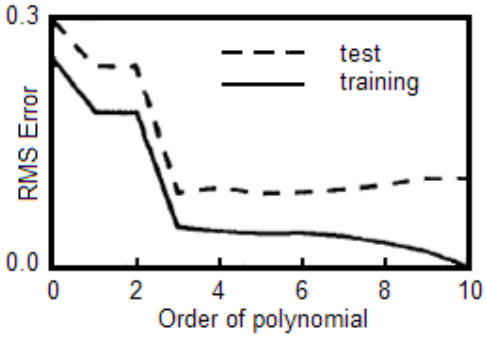


Figure 2.4 RMS error as a function of the order of the polynomial for both training and test sets

when the number of degrees of freedom in the model is relatively small compared to the size of the data set (4 free parameters for M = 3, compared with 11 data points in this example).

2.3 Model complexity

Using an example of polynomial curve fitting, we have seen that the best generalization performance is achieved by a model whose complexity (measured here by the order of the polynomial) is neither too small nor too large [3]. The problem of finding the optimal complexity for a model provides an example of Occam's razor, named after William of Occam (1285 -13 49). This is the principle that we should prefer simpler models to more complex models, and that this preference should be traded off against the extent to which the models fit the data. Thus a highly complex model which fits the data extremely well (such as the l0th-order polynomial above) actually gives a poorer representation of the systematic aspects of the data than would a simpler model (such as the 3-order polynomial). A model which is too simple, however, as in the 1st-order polynomial, is also not preferred as gives too poor a fit to the data.

An alternative approach to optimizing the generalization performance of a model is to control its effective complexity. This can be achieved by considering a model with many adjustable parameters, and then altering the training procedure by adding a penalty term Ω to the error function. The total error then becomes

$$\tilde{E} = E + \nu\,\Omega \tag{2.9}$$

where Ω is called a regularization term. The value of Ω depends on the mapping function *y(x)*, and if the functional form of Ω is chosen appropriately, it can be used to control over-fitting. For example, if we examine the function represented by the l0th-order polynomial in Fig. 2.3, we see that it has large oscillations, and hence the function *y(x)* has regions of huge curvature. We might therefore choose a regularization function which is large for functions with large values of the second derivative, such, as

$$\Omega = \frac{1}{2}\int \left(\frac{d^2y}{dx^2}\right)^2 dx \tag{2.10}$$

The parameter ν in (2.9) controls the extent to which the regularization term influences the form of the solution, and hence controls the effective complexity of the model.

We have seen that, for a fixed size of data set, it is important to achieve the optimum level of complexity for the model in order to minimize the combination of bias and variance. By using a sequence of successively larger data sets, however, and a corresponding set of models with successively greater complexity, it is possible in principle to reduce both bias and variance simultaneously and hence to improve the generalization performance. The ultimate generalization achievable will be limited by the intrinsic noise on the data.

The next chapter presents basic concepts of approximation using Radial Basis Functions.

3 RADIAL BASIS FUNCTIONS

A radial basis function (RBF) is a real-valued function whose value depends only on the distance from the origin, so that

$$\phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|) \tag{3.1}$$

or alternatively on the distance from some other point *c*, called a *center*, so that

$$\phi(\boldsymbol{x}, \boldsymbol{c}) = \phi(\|\boldsymbol{x} - \boldsymbol{c}\|) \tag{3.2}$$

Any function φ that satisfies the property $\phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|)$ is a radial function [4]. The norm is usually Euclidan distance, although other distance functions are also possible. For example by using Lukaszyk-Karmowski metric it is for some radial functions possible [5] to avoid problems with ill conditioning of the matrix solved to determine coefficients w_i (see below), since the $\|\boldsymbol{x}\|$ is always greater than zero.

Sums of radial basis functions are typically used to approximate given functions. This approximation process can also be interpreted as a simple kind of neural network.

3.1 RBF types

Commonly used types of radial basis functions (writing $r = \|x - c_i\|$) are:

Gaussian:

$$\varphi(r) = e^{-\beta r^2} \qquad \text{for some } \beta > 0 \tag{3.3}$$

Multiquadratic:

$$\varphi(r) = \sqrt{r^2 + \beta^2} \quad \text{for some } \beta > 0 \tag{3.4}$$

Polyharmonic spline:

$$\varphi(r) = r^k, k = 1, 3, 5 \ldots \tag{3.5}$$

$$\varphi(r) = r^k \ln(r), k = 2, 4, 6, \ldots \tag{3.6}$$

Thin plate spline:

$$\varphi(r) = r^2 \ln(r) \tag{3.7}$$

3.2 Approximation

Radial basis functions are typically used to build up function approximations of the form

$$y(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi(\|\boldsymbol{x} - \boldsymbol{c_i}\|) \tag{3.8}$$

where the approximating function $y(\boldsymbol{x})$ is represented as a sum of N radial basis functions, each associated with a different center $\boldsymbol{c_i}$, and weighted by an appropriate coefficient w_i. The weights w_i can be estimated using the matrix methods of linear least squares, because the approximating function is *linear* in the weights.

Approximation schemes of this kind have been particularly used in time series prediction and control of nonlinear systems exhibiting sufficiently simple chaotic behavior, 3D reconstruction in computer graphics (for example, hierarchical RBF).

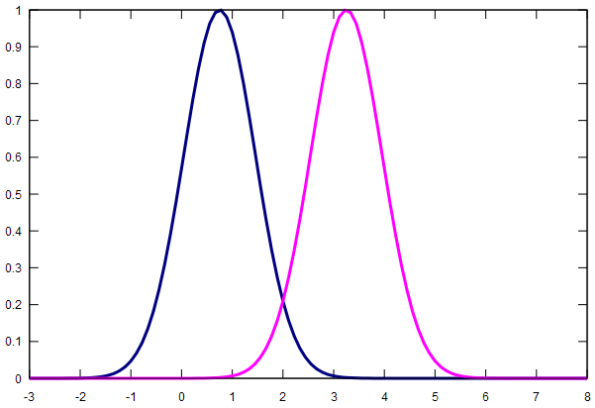


Figure 3.1 Unnormalized Radial Basis Functions

3.3 Exact interpolation

Radial basis function methods have their origins in techniques for performing exact interpolation of a set of data points in a multi-dimensional space [3, 6]. The exact interpolation problem requires every input vector to be mapped exactly onto the corresponding target vector, and forms a convenient starting point for the discussion of radial basis function networks.

Consider a mapping from a d-dimensional input space **x** to a one-dimensional target space t. The data set consists of N input vectors $\mathbf{x}^n$, together with corresponding targets t^n. The goal is to find a function $y(\boldsymbol{x})$ such that

$$y(x) = t^n, \qquad n = 1, \ldots, N \tag{3.9}$$

The radial basis function approach [6] introduces a set of N *basis functions*, one for each data point, which take the form $\phi((||\mathbf{x} - \boldsymbol{c_i}||)$ where ϕ (•) is some non-linear function whose form will be discussed shortly. Thus the i-th such function depends on the distance

$\|\mathbf{x} - \boldsymbol{c}_i\|$, usually taken to be Euclidean, between $\boldsymbol{x}$ and $\boldsymbol{c}_i$. The output of the mapping is then taken to be a linear combination of the basis functions

$$y(\boldsymbol{x}) = \sum_n w_n \phi(\|\boldsymbol{x} - \boldsymbol{c_i}\|) \tag{3.10}$$

We recognize this as having the same form as the generalized linear discriminant function

$$y_k(\boldsymbol{x}) = \sum_{j=0}^{M} w_{kj} \phi_j(\boldsymbol{x}) \tag{3.11}$$

The interpolation conditions (3.9) can then be written in matrix form as

$$\boldsymbol{\Phi}\, \boldsymbol{w} = \boldsymbol{t} \tag{3.12}$$

where $\boldsymbol{t} \equiv (t^{\boldsymbol{n}}), w \equiv (w_n)$, and the square matrix $\boldsymbol{\Phi}$ has elements $\boldsymbol{\Phi_{nn'}} = \boldsymbol{\phi}\left(\left\|x^{\boldsymbol{n}} - \boldsymbol{x^{n'}}\right\|\right)$. Provided the inverse matrix $\boldsymbol{\Phi^{-1}}$ exists we can solve (3.12) to give

$$\boldsymbol{w} = \boldsymbol{\Phi^{-1}} \boldsymbol{t} \tag{3.13}$$

It has been shown [7] that, for a large class of functions ϕ (•), the matrix $\boldsymbol{\Phi}$ is indeed non-singular provided the data points are distinct. When the weights in (3.10) are set to the values given by (3.12), the function *y(x)* represents a continuous differentiable surface which passes exactly through each data point.

Both theoretical and empirical studies [6] show that, in the context of the exact interpolation problem, many properties of the interpolating function are relatively insensitive to the precise form of the non-linear function ϕ (•). Several forms of basis function have been considered, the most common being the Gaussian

$$\phi(x) = \mathrm{e}^{\left(-\frac{x^2}{2\sigma^2}\right)} \tag{3.14}$$

where σ is a parameter whose value controls the smoothness properties of the interpolating function. The Gaussian (3.14) is a *localized* basis function with the property that $\phi \to 0$ as $|x| \to \infty$. Another choice of basis function with the same property is the function

$$\phi(x) = (x^2 + \sigma^2)^{-\alpha}, \qquad \alpha > 0 \tag{3.15}$$

It is not, however, necessary for the functions to be localized, and other possible choices are the thin-plate spline function

$$\phi(x) = x^2 \ln(x) \tag{3.16}$$

the function

$$\phi(x) = (x^2 + \sigma^2)^{\beta}, \qquad 0 < \beta < 1 \tag{3.17}$$

which for $\beta = 1/2$ is known as the multi-quadric function, the cubic

$$\phi(x) = x^3 \tag{3.18}$$

and the 'linear' function

$$\phi(x) = x \tag{3.19}$$

which all have the property that $\phi \to \infty$ as $x \to \infty$.

Note that (3.19) is linear in $x = \|\mathbf{x} - \mathbf{x}^n\|$ and so is still a non-linear function of the components of **x**. In one dimension, it leads to a piecewise-linear interpolating function which represents the simplest form of exact interpolation. We shall focus most of our attention on Gaussian basis functions since, as well as being localized, they have a number of useful analytical properties.

The generalization to several output variables is straightforward. Each input vector x^nmust be mapped exactly onto an output vector t^n having components t_k^n so that (3.9) becomes

$$y_k(\boldsymbol{x}^n) = \boldsymbol{t}_k^n, \qquad n = 1, \ldots, N \tag{3.20}$$

where the $y_k(\boldsymbol{x})$ are obtained by linear superposition of the same *N* basis functions as used for the single-output case

$$y_k(\boldsymbol{x}^n) = \sum_n w_{kn}\phi(\|\boldsymbol{x} - \boldsymbol{x}^n\|) \tag{3.21}$$

The weight parameters are obtained by analogy with (3.13) in the form

$$w_{kn} = \sum_{n'}(\Phi^{-1})_{nn'}\, t_k^{n'} \tag{3.22}$$

Note that in (3.22) the same matrix Φ^{-1} is used for each of the output functions.

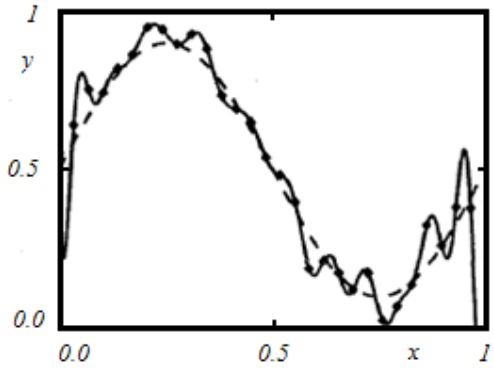


Figure 3.2 A simple example of exact interpolation using radial basis functions.

A set of 30 data points in Fig. 3.2 was generated by sampling the function $y = 0.5 + 0.4\sin(2\pi x)$, shown by the dashed curve, and adding Gaussian noise with standard deviation 0.05. The solid curve shows the interpolating function which results from using Gaussian basis functions of the form (3.14) with width parameter $\sigma = 0.0067$ which corresponds to roughly twice the spacing of the data points. Values for the weights were found using matrix inversion techniques as discussed in the text.

The following chapter outlines estimation with the least squares method.

4 REAL-TIME PARAMETER ESTIMATION

On-line determination of process parameters is a key element in adaptive control [8]. A recursive parameter estimator appears explicitly as a component of a self-tuning regulator. Parameter estimation also occurs implicitly in a model-reference adaptive controller. This chapter presents some methods for real-time parameter estimation. It is useful to view parameter estimation in the broader context of system identification. The key elements of system identification are selection of model structure, experiment design, parameter estimation, and validation. Since system identification is executed automatically in adaptive systems, it is essential to have a good understanding of all aspects of the problem. Selection of model structure and parameterization are fundamental issues. Simple transfer function models will be used in this chapter. The identification problems are simplified significantly if the models are linear in the parameters.

The experiment design is crucial for successful system identification. In control problems this boils down to selection of the input signal. Choosing an input signal requires some knowledge of the process and the intended use of the model. In adaptive systems there is an additional complication because the input signal to the plant is generated by feedback. In certain cases this does not permit the parameters to be determined uniquely, a situation that has far-reaching consequences. In some cases it may be necessary to introduce perturbation signals. In adaptive control the parameters of a process change continuously, so it is necessary to have estimation methods that update the parameters recursively.

In solving identification problems it is very important to validate the results. This is especially important for adaptive systems, in which identification is done automatically

The least-squares method is a basic technique for parameter estimation. The method is particularly simple if the model has the property of being *linear in the parameters,* in this case the least-squares estimate can be calculated analytically. A compact presentation of the method of least squares is given in this chapter. The formulas for the estimate are derived and statistical interpretations are given.

4.1 Least squares and regression models

Karl Friedrich Gauss formulated the principle of least squares at the end of the eighteenth century and used it to determine the orbits of planets and asteroids. Gauss stated that, according to this principle, the unknown parameters of a mathematical model should be chosen in such a way that the sum of the squares of the differences between the actually observed and the computed values, multiplied by numbers that measure the degree of precision, is a minimum. The least squares method can be applied to large variety of problems. It is particularly simple for a mathematical model that can be written in the form

$$y(i) = \varphi_1(i)\theta_1^0 + \varphi_2(i)\theta_2^0 + \cdots + \varphi_n(i)\theta_n^0 = \varphi^T(i)\theta^0 \tag{4.1}$$

Where y is the observed variable, $\theta_1^0, \theta_2^0, \ldots, \theta_n^0$ are parameters of the model to be determined, and $\varphi_1, \varphi_2, \ldots, \varphi_n$ are known functions that may depend on other known variables. The vectors

$$\boldsymbol{\varphi}^T(i) = (\varphi_1(i) \quad \varphi_2(i) \quad \ldots \quad \varphi_n(i)) \tag{4.2}$$

$$\boldsymbol{\theta^0} = (\theta_1^0 \quad \theta_2^0 \quad \ldots \quad \theta_n^0)^T \tag{4.3}$$

have also been introduced. The model is indexed by the variable *i*, which often denotes time. It will be assumed initially that the index set is a discrete set. The variables φ_i are called the regression variables or the regressors, and the model in Eq. (4.1) is also called a regression model. Pairs of observations and regressors $\{(y(i), \varphi(i)), i = 1, 2, \ldots, t\}$ are obtained from an experiment. The problem is to determine the parameters in such a way that the outputs computed from the model in Eq. (4.1) agree as closely as possible with the measured variables *y(i)* in the sense of least squares. That is, the parameter θ should be chosen to minimize the least-squares loss function

$$V(\theta, t) = \frac{1}{2}\sum_{i=1}^{t}(y(i) - \varphi^T(i)\theta)^2 \tag{4.4}$$

Since the measured variable y is linear in parameters $\boldsymbol{\theta^0}$ and the least-squares criterion is quadratic, the problem admits an analytical solution. Introduce the notations

$$Y(t) = (\ y(1) \quad y(2) \quad \ldots \quad y(t)\ )^T \tag{4.5}$$

$$\mathrm{E}(t) = (\ \varepsilon(1) \quad \varepsilon(2) \quad \ldots \quad \varepsilon(t)\ )^T \tag{4.6}$$

$$\boldsymbol{\Phi}(t) = \begin{pmatrix} \varphi^T(1) \\ \vdots \\ \varphi^T(t) \end{pmatrix} \tag{4.7}$$

$$P(t) = (\Phi^{\mathrm{T}}(t)\Phi(t))^{-1} = (\sum_{i=1}^{t}\varphi(i)\varphi^T(i))^{-1} \tag{4.8}$$

where the $residuals\ e(i)$ are defined by

$$\varepsilon(i) = y(i) - \hat{y}(i) = y(i) - \varphi^T(i)\ \theta \tag{4.9}$$

With these notations the loss function (4.4) can be written as

$$V(\theta, t) = \frac{1}{2}\sum_{i=1}^{t}\left(\varepsilon^2(i)\right) \approx \frac{1}{2}\mathrm{E}^{\mathrm{T}}\mathrm{E} = \frac{1}{2}\|\mathrm{E}\|^2 \tag{4.10}$$

where $\|\mathrm{E}\|$ can be written as

$$\|\mathrm{E}\| = \mathrm{Y} - \widehat{\mathrm{Y}} = \mathrm{Y} - \Phi\theta \tag{4.11}$$

The solution to the least-squares problem is given by the following theorem.

Theorem 4.1 Least-squares estimation:

The function of Eq. (4.4) is minimal for parameters $\hat{\theta}$ such that

$$\Phi^T \Phi \hat{\theta} = \Phi^{\mathrm{T}} \mathbf{Y} \tag{4.12}$$

If the matrix $\Phi^T \Phi$ is nonsingular, the minimum is unique and given by

$$\widehat{\boldsymbol{\theta}} = (\Phi^{\mathrm{T}} \Phi)^{-1} \Phi^{\mathrm{T}} \boldsymbol{\theta^0} \tag{4.13}$$

4.2 Statistical Interpretation

The least-squares method can be interpreted in statistical terms. It is then necessary to make assumptions about how the data has been generated [8]. Assume, that the process is

$$y(i) = \varphi^T(i) \boldsymbol{\theta}^0 + \boldsymbol{e}(i) \tag{4.14}$$

where $\boldsymbol{\theta}^0$ is the vector of "true" parameters and *{e(i), i = 1, 2, …}* is a sequence of independent, equally distributed random variables with zero mean. It is also assumed that *e* is independent of φ. Equation (4.4) can be written as

$$\boldsymbol{Y} = \Phi \boldsymbol{\theta}^0 + \mathbf{E} \tag{4.15}$$

Multiplying by $(\Phi^{\mathrm{T}} \Phi)^{-1} \Phi^{\mathrm{T}}$ gives

$$(\Phi^{\mathrm{T}} \Phi)^{-1} \Phi^{\mathrm{T}} \mathbf{Y} = \hat{\theta} = \hat{\theta}^0 + (\Phi^{\mathrm{T}} \Phi)^{-1} \Phi^{\mathrm{T}} \mathbf{E} \tag{4.16}$$

Provided that $\mathbf{E}$ is independent of Φ^{T}, which is equivalent to saying that *e(i)* is independent of $\varphi(i)$, the mathematical expectation of $\hat{\theta}$ is equal to $\hat{\theta}^0$. An estimate with this property is called unbiased. The following theorem is given without proof.

Theorem 4.2 Statistical properties of least-squares estimation:

Consider the estimate in Eq. (4.13) and assume that data is generated from Eq. (4.14), where *{e(i), i = 1, 2, ...}* is a sequence of independent random variables with zero mean and variance σ^2. Let E denote mathematical expectation and *cov* the covariance of a random variable.

If $\Phi^{\mathrm{T}} \Phi$ is nonsingular, then

(i) $\mathrm{E}\, \hat{\theta}(\mathrm{t}) = \theta^0$

(ii) $cov\, \hat{\theta}(\mathrm{t}) = \theta^0$

(iii) $\sigma^2(t) = 2V(\hat{\theta}, t)/(t - n)$ is an unbiased estimate of σ^2

where *n* is the number of parameters in θ^0 and $\hat{\theta}$ and *t* is the number of data points. □

The theorem states that the estimates are unbiased, that is $\mathrm{E}\, \hat{\theta}(\mathrm{t}) = \theta^0$. Further, it is desirable that an estimate converge to the true parameter value as the number of observations

increases toward infinity. This property is called *consistency* [9]. There are several notions of consistency corresponding to different convergence concepts for random variables. Mean square convergence is one possibility, which can be investigated simply by analyzing the variance of the estimate. The result (ii) can be used to determine how the variance of the estimate decreases with the number of observations.

The modeling terminology and certain examples of modeling with least squares method will be presented in the next chapter.

5 STATISTICAL MODEL

Basic 1D statistical model [13]

$$y = g(x) + \epsilon, x \in [a, b] \tag{5.1}$$

where ϵ is independent identically distributed $N(0, \sigma_\epsilon^2)$

g(x) is representation by basis functions $\phi_j(x)$

$$y(x, \boldsymbol{w}) = \sum_{j=1}^{m} w_j \phi_j(x) + \epsilon \tag{5.2}$$

The random part of the measurements ϵ are independent identically distributed $N(0, \sigma_\epsilon^2)$ random variables with the notation for column vectors $\boldsymbol{\phi}(x)$ and *w*

$$\boldsymbol{\phi}^T(x) = \big(\phi_1(x), \phi_2(x), \dots, \phi_m(x)\big), x \in \mathbb{R} \tag{5.3}$$

n is number of data points

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \tag{5.4}$$

and let's denote the vector of y-values $\boldsymbol{Y}^T = (y_1, y_2, \dots, y_n)$

given the least squares estimate $\widehat{\boldsymbol{w}}$ of $\boldsymbol{w}$ is

$$\widehat{\boldsymbol{w}} = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\mathbf{Y} \tag{5.5}$$

Where Φ is $n \times m$ matrix with entries $\Phi(i, j) = \phi_j(x_i)$.

5.1 Statistical model with linear equations

Assumptions:
The number of data points *n = 4*,
the number of basis functions *m = 2*,
data points $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$,
the vector of *y*-values $\boldsymbol{Y}^T = (y_1, y_2, y_3, y_4)$
the vector of unknown estimates $\widehat{\boldsymbol{w}}^T = (\widehat{w}_1, \widehat{w}_2)$

$$y(x) = w_1\phi_1(x) + w_2\phi_2(x) \tag{5.6}$$

$$\Phi\widehat{\boldsymbol{w}}^T = \boldsymbol{Y} \tag{5.7}$$

$$\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) \\ \phi_1(x_2) & \phi_2(x_2) \\ \phi_1(x_3) & \phi_2(x_3) \\ \phi_1(x_4) & \phi_2(x_4) \end{bmatrix} \begin{bmatrix} \widehat{w}_1 \\ \widehat{w}_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \tag{5.8}$$

5.2 The least-squares method

The number of data points $n = 4$, the number of basis functions $m = 3$

$$\widehat{\boldsymbol{w}} = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\boldsymbol{Y} \tag{5.9}$$

The matrix product

$$\Phi^{\mathrm{T}}\Phi = \begin{bmatrix} \phi_1(x_1) & \phi_1(x_2) & \phi_1(x_3) & \phi_1(x_4) \\ \phi_2(x_1) & \phi_2(x_2) & \phi_2(x_3) & \phi_2(x_4) \\ \phi_3(x_1) & \phi_3(x_2) & \phi_3(x_3) & \phi_3(x_4) \end{bmatrix} \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) \\ \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) \end{bmatrix} \tag{5.10}$$

Radial basis functions in the form of

$$\phi(x) = [\phi_1(x) \quad \phi_2(x) \quad \phi_3(x)] = [e^{\frac{-(x-c_1)^2}{s_1}}, e^{\frac{-(x-c_2)^2}{s_2}}, e^{\frac{-(x-c_3)^2}{s_3}}] \tag{5.11}$$

Note the normal (or Gaussian) distribution in the form:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}} \tag{5.12}$$

Properties of variance and its derivation follows in the next chapter.

6 VARIANCE

The least squares solution $\widehat{w}$ is unbiased [10, 15]

$$E\widehat{w} = w \tag{6.1}$$
$$var(\widehat{w}) = \sigma_\epsilon^2(\Phi^{\mathrm{T}}\Phi)^{-1} \tag{6.2}$$

Then the predicted value y_p of y at some $x = s$ is

$$y_p(s) = \widehat{w}^T\phi(s) \tag{6.3}$$

while its actual value is given as

$$y_a(s) = w^T\phi(s) + \epsilon \tag{6.4}$$

Consequently

$$E\left(y_p(s) - y_a(s)\right) = 0 \tag{6.5}$$

$$var\left(y_p(s) - y_a(s)\right) = \phi^T(s)var(\widehat{w})\phi(s) + \sigma_\epsilon^2 \tag{6.6}$$

We can see that the variance of prediction depends on measurement locations used for model estimation:

$$\sigma^2(s) = \phi^T(s)A^{-1}\phi(s) + \sigma_\epsilon^2 \tag{6.7}$$

$$A^{-1} = \sigma_\epsilon^2(\Phi^{\mathrm{T}}\Phi)^{-1} \tag{6.8}$$

Where $A = A(n)$ is determined by n measurements at locations $x_1, x_2, \dots, x_n$ and by the choice of basis functions ϕ.

A can be rewritten in the incremental form:

$$A = \frac{1}{\sigma_\epsilon^2}\sum_{i=1}^{n}\phi\left(x^i\right)\phi\left(x^i\right)^T \tag{6.9}$$

from which it is visible how the variance itself varies with the measurement points distribution. The fact that we can control the width of the variance can be utilized for the stopping criterion definition.

6.1 Variance derivation

1. From the fact that the random vector X can be transformed [15] by a coefficient matrix B

$$var(BX) = B.var(X)B^T \tag{6.10}$$

2. The least-squares (LS) solution has several important properties that are used in the proof of the variance formula:

$$var(\widehat{w}) = \sigma_\epsilon^2(\Phi^{\mathrm{T}}\Phi)^{-1} \tag{6.11}$$

Proof:
We start with the LS solution of the basic model

$$\Phi^{\mathrm{T}}\Phi\widehat{\mathrm{W}} = \Phi^{\mathrm{T}}\mathrm{Y} \tag{6.12}$$

$$\widehat{w} = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\mathrm{Y} \tag{6.13}$$

The variance formula for transformed random variables X says

$$var(BX) = B\big(var(X)\big)B^T \tag{6.14}$$

Therefore [10]

$$var(\widehat{w}) = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\, var(Y)((\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}})^T \tag{6.15}$$

Using the fact that the measurement errors are independent

$$var(Y) = \sigma_\epsilon^2 I \tag{6.16}$$

The variance formula can be simplified

$$var(\widehat{w}) = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\sigma_\epsilon^2 I((\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}})^T \tag{6.17}$$

$$var(\widehat{w}) = \sigma_\epsilon^2(\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}((\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}})^{\mathrm{T}} \tag{6.18}$$

Because $(AB)^T = B^T A^T$ and $(A^T)^T = A$

$$((\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}})^T = (\Phi(\Phi^{\mathrm{T}}\Phi)^{-1})^T \tag{6.19}$$

$$var(\widehat{w}) = \sigma_\epsilon^2 I(\Phi^{\mathrm{T}}\Phi)^{-1}(\Phi^{\mathrm{T}}\Phi)((\Phi^{\mathrm{T}}\Phi)^{-1})^T \tag{6.20}$$

and further $(A^{-1})^T = (A^T)^{-1}$ implies

$$((\Phi^{\mathrm{T}}\Phi)^{-1})^T = (\Phi^{\mathrm{T}}\Phi)^{-1} \tag{6.21}$$

and finally

$$var(\widehat{w}) = \sigma_\epsilon^2(\Phi^{\mathrm{T}}\Phi)^{-1}(\Phi^{\mathrm{T}}\Phi)(\Phi^{\mathrm{T}}\Phi)^{-1} \tag{6.22}$$

$$var(\widehat{w}) = \sigma_\epsilon^2(\Phi^{\mathrm{T}}\Phi)^{-1} \tag{6.23}$$

6.2 General variance for two basis functions

The number of data points *n*, the number of basis functions *m* = 2:

$$\phi(x) = [\phi_1(x), \phi_2(x)] = [1, x] \tag{6.24}$$

$$\Phi^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \end{bmatrix} \tag{6.25}$$

The variance

$$var(x) = \phi(x)(\Phi^{\mathrm{T}}\Phi)^{-1}\phi(x)^T \tag{6.26}$$

$$var(x) = [1, x](\Phi^{\mathrm{T}}\Phi)^{-1}[1, x]^T \tag{6.27}$$

The matrix product

$$\Phi^{\mathrm{T}}\Phi = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \tag{6.28}$$

And its inverse (using the determinant formula):

$$(\Phi^{\mathrm{T}}\Phi)^{-1} = \begin{bmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & n \end{bmatrix} \frac{1}{n\sum x_i^2 - (\sum x_i)^2} \tag{6.29}$$

From equations (1) and (2)

$$var(x) = \frac{1}{n} + \frac{(x-\bar{x})^2}{\sum x_i^2 - n\bar{x}^2} \tag{6.30}$$

Or equivalently

$$var(x) = \frac{1}{n} + \frac{(x-\bar{x})^2}{\sum (x_i-\bar{x})^2} \tag{6.31}$$

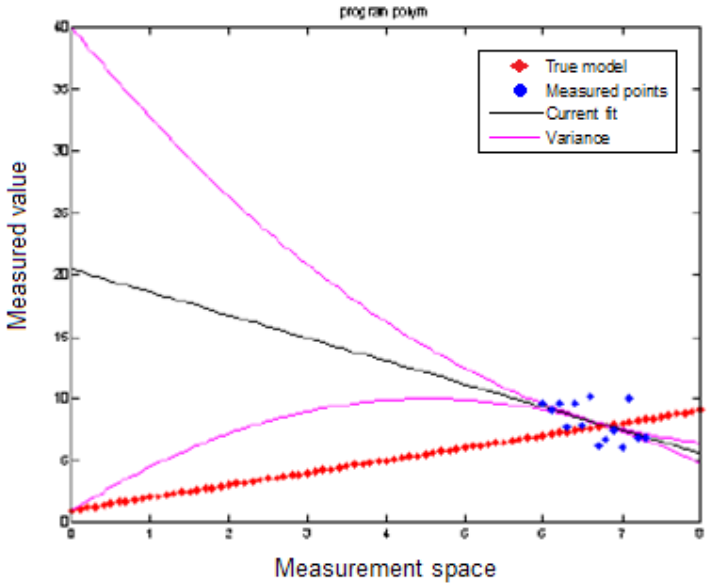


Figure 6.1 Example of a fit with insufficient number of data points

where $\bar{x} = \frac{1}{n}\sum x_i$ and the following equations have been utilized:

$$\sum x_i^2 - n\bar{x}^2 = \sum(x_i^2) - 2n\bar{x}^2 + n\bar{x}^2 \tag{6.32}$$

$$\sum(x_i^2) - 2\sum x_i\bar{x} + \sum \bar{x}^2 = \sum(x_i^2 - 2x_i\bar{x} + \bar{x}^2) = \sum(x_i - \bar{x})^2 \tag{6.33}$$

The *var(x)* formula in (6.31) implies that for conveniently placed measurement points x_i variance converges to zero (see denominator). It also implies that the magnitude of *var(x)* depends on the placement of the measurement points [13].

The Fig. 6.1 shows the example of a situation where the true model was approximated with insufficient number of data points in one sub-region of the measurement area.

The next chapter deals with introduction to genetic algorithms.

7 GENETIC ALGORITHMS

7.1 The appeal of evolution

To evolutionary-computation researchers, the mechanisms of evolution seem well suited for some of the most pressing computational problems in many fields [16]. Many computational problems require searching through a huge number of possibilities for solutions. One example is the problem of computational protein engineering, in which an algorithm is sought that will search among the vast number of possible amino acid sequences for a protein with specified properties. Another example is searching for a set of rules or equations that will predict the ups and downs of a financial market, such as that for foreign currency. Such search problems can often benefit from an effective use of parallelism, in which many different possibilities are explored simultaneously in an efficient way. For example, in searching for proteins with specified properties, rather than evaluate one amino acid sequence at a time it would be much faster to evaluate many simultaneously. What is needed is both computational parallelism (i.e., many processors evaluating sequences at the same time) and an intelligent strategy for choosing the next set of sequences to evaluate.

Many computational problems require a computer program to be *adaptive*—to continue to perform well in a changing environment. This is typified by problems in robot control in which a robot has to perform a task in a variable environment, and by computer interfaces that must adapt to the idiosyncrasies of different users. Other problems require computer programs to be innovative—to construct something truly new and original, such as a new algorithm for accomplishing a computational task or even a new scientific discovery. Finally, many computational problems require complex solutions that are difficult to program by hand. A striking example is the problem of creating artificial intelligence. Early on, AI practitioners believed that it would be straightforward to encode the rules that would confer intelligence on a program; expert systems were one result of this early optimism. Nowadays, many AI researchers believe that the "rules" underlying intelligence are too complex for scientists to encode by hand in a "top-down" fashion. Instead they believe that the best route to artificial intelligence is through a "bottom-up" paradigm in which humans write only very simple rules, and complex behaviors such as intelligence emerge from the massively parallel application and interaction of these simple rules. Connectionism (i.e., the study of computer programs inspired by neural systems) is one example of this philosophy [18]; evolutionary computation is another. In connectionism the rules are typically simple "neural" thresholding, activation spreading, and strengthening or weakening of connections; the hoped-for emergent behavior is sophisticated pattern recognition and learning. In evolutionary computation the rules are typically "natural selection" with variation due to crossover and/or mutation; the hoped-for emergent behavior is the design of high-quality solutions to difficult problems and the ability to adapt these solutions in the face of a changing environment.

Biological evolution is an appealing source of inspiration for addressing these problems. Evolution is, in effect, a method of searching among an enormous number of possibilities for "solutions." In biology the enormous set of possibilities is the set of possible genetic sequences, and the desired "solutions" are highly fit organisms—organisms well able to survive and reproduce in their environments. Evolution can also be seen as a method for *designing* innovative solutions to complex problems. For example, the mammalian immune system is a marvelous evolved solution to the problem of germs invading the body. Seen in

this light, the mechanisms of evolution can inspire computational search methods. Of course the fitness of a biological organism depends on many factors—for example, how well it can weather the physical characteristics of its environment and how well it can compete with or cooperate with the other organisms around it. The fitness criteria continually change as creatures evolve, so evolution is searching a constantly changing set of possibilities. Searching for solutions in the face of changing conditions is precisely what is required for adaptive computer programs. Furthermore, evolution is a massively parallel search method: rather than work on one species at a time, evolution tests and changes millions of species in parallel. Finally, viewed from a high level the "rules" of evolution are remarkably simple: species evolve by means of random variation (via mutation, recombination, and other operators), followed by natural selection in which the fittest tend to survive and reproduce, thus propagating their genetic material to future generations. Yet these simple rules are thought to be responsible, in large part, for the extraordinary variety and complexity we see in the biosphere.

7.2 Biological terminology

At this point it is useful to formally introduce some of the biological terminology that will be used throughout the book. In the context of genetic algorithms, these biological terms are used in the spirit of analogy with real biology, though the entities they refer to are much simpler than the real biological ones.

All living organisms consist of cells, and each cell contains the same set of one or more *chromosomes*—strings of DNA—that serve as a "blueprint" for the organism. A chromosome can be conceptually divided into *genes*— each of which encodes a particular protein. Very roughly, one can think of a gene as encoding a *trait,* such as eye color. The different possible "settings" for a trait (e.g., blue, brown, hazel) are called *alleles.* Each gene is located at a particular *locus* (position) on the chromosome.

Many organisms have multiple chromosomes in each cell. The complete collection of genetic material (all chromosomes taken together) is called the organism's *genome.* The term *genotype* refers to the particular set of genes contained in a genome. Two individuals that have identical genomes are said to have the same genotype. The genotype gives rise, under fetal and later development, to the organism's *phenotype*—its physical and mental characteristics, such as eye color, height, brain size, and intelligence.

Organisms whose chromosomes are arrayed in pairs are called *diploid;* organisms whose chromosomes are unpaired are called *haploid.* In nature, most sexually reproducing species are diploid, including human beings, who each have 23 pairs of chromosomes in each somatic (non-germ) cell in the body. During sexual reproduction, *recombination* (or *crossover)* occurs: in each parent, genes are exchanged between each pair of chromosomes to form a *gamete* (a single chromosome), and then gametes from the two parents pair up to create a full set of diploid chromosomes. In haploid sexual reproduction, genes are exchanged between the two parents' single-strand chromosomes. Offspring are subject to *mutation,* in which single nucleotides (elementary bits of DNA) are changed from parent to offspring, the changes often resulting from copying errors. The *fitness* of an organism is typically defined as the probability that the organism will live to reproduce (*viability)* or as a function of the number of offspring the organism has (*fertility*).

In genetic algorithms, the term *chromosome* typically refers to a candidate solution to a problem, often encoded as a bit string. The "genes" are either single bits or short blocks of adjacent bits that encode a particular element of the candidate solution (e.g., in the context of multi parameter function optimization the bits encoding a particular parameter might be considered to be a gene). An allele in a bit string is either 0 or 1; for larger alphabets more alleles are possible at each locus. Crossover typically consists of exchanging genetic material between two single chromosome haploid parents. Mutation consists of flipping the bit at a randomly chosen locus (or, for larger alphabets, replacing a the symbol at a randomly chosen locus with a randomly chosen new symbol).

Most applications of genetic algorithms employ haploid individuals, particularly, single-chromosome individuals. The genotype of an individual in a GA using bit strings is simply the configuration of bits in that individual's chromosome. Often there is no notion of "phenotype" in the context of GAs, although more recently many workers have experimented with GAs in which there is both a genotypic level and a phenotypic level (e.g., the bit-string encoding of a neural network and the neural network itself).

7.3 Search spaces and fitness landscape

The idea of searching among a collection of candidate solutions for a desired solution is so common in computer science that it has been given its own name: searching in a "search space."[16] Here the term "search space" refers to some collection of candidate solutions to a problem and some notion of "distance" between candidate solutions. For an example, let us take one of the most important problems in computational bioengineering: the aforementioned problem of computational protein design. Suppose you want use a computer to search for a protein—a sequence of amino acids—that folds up to a particular three-dimensional shape so it can be used, say, to fight a specific virus. The search space is the collection of all possible protein sequences—an infinite set of possibilities. To constrain it, let us restrict the search to all possible sequences of length 100 or less—still a huge search space, since there are 20 possible amino acids at each position in the sequence. (How many possible sequences are there?) If we represent the 20 amino acids by letters of the alphabet, candidate solutions will look like this:

A G G M C G B L....

We will define the distance between two sequences as the number of positions in which the letters at corresponding positions differ. For example, the distance between A G G M C G B L and MG G M C G B L is 1, and the distance between A G G M C G B L and L B M P A F G A is 8. An algorithm for searching this space is a method for choosing which candidate solutions to test at each stage of the search. In most cases the next candidate solution(s) to be tested will depend on the results of testing previous sequences; most useful algorithms assume that there will be some correlation between the quality of "neighboring" candidate solutions—those close in the space. Genetic algorithms assume that high-quality "parent" candidate solutions from different regions in the space can be combined via crossover to, on occasion, produce high-quality "offspring" candidate solutions.

Another important concept is that of "fitness landscape." Originally defined by the biologist Sewell Wright (1931) in the context of population genetics, a fitness landscape is a representation of the space of all possible genotypes along with their fitnesses.

Suppose, for the sake of simplicity, that each genotype is a bit string of length *l,* and that the distance between two genotypes is their "Hamming distance"—the number of locations at which corresponding bits differ. Also suppose that each genotype can be assigned a real-valued fitness. A fitness landscape can be pictured as an *(l* + 1)-dimensional plot in which each genotype is a point in *l* dimensions and its fitness is plotted along the (l + 1)st axis. A simple landscape for *l* = 2 is shown in Fig 7.1. Such plots are called landscapes because the plot of fitness values can form "hills," "peaks," "valleys," and other features analogous to those of physical landscapes. Under Wright's formulation, evolution causes populations to move along landscapes in particular ways, and "adaptation" can be seen as the movement toward local peaks. (A "local peak," or "local optimum," is not necessarily the highest point in the landscape, but any small

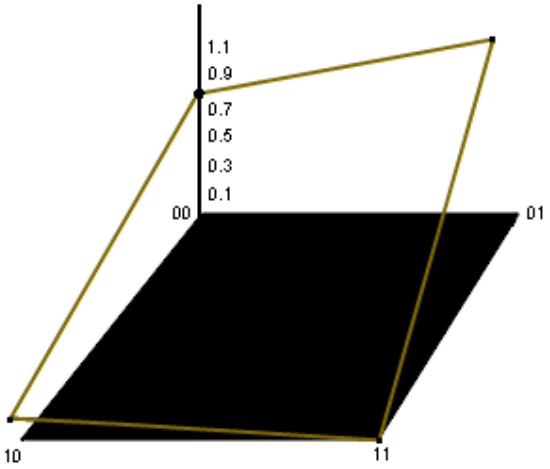


Figure 7.1: A simple fitness landscape for l = 2. Here f(00) = 0.7, f(01) = 1.0, f(10) = 0.1, and f(11) = 0.0.

movement away from it goes downward in fitness.) Likewise, in GAs the operators of crossover and mutation can be seen as ways of moving a population around on the landscape defined by the fitness function.

The idea of evolution moving populations around in unchanging landscapes is biologically unrealistic for several reasons. For example, an organism cannot be assigned a fitness value independent of the other organisms in its environment; thus, as the population changes, the fitnesses of particular genotypes will change as well. In other words, in the real world the "landscape" cannot be separated from the organisms that inhabit it. In spite of such caveats, the notion of fitness landscape has become central to the study of genetic algorithms, and it will come up in various guises throughout this book.

7.4 Elements of genetic algorithms

It turns out that there is no rigorous definition of "genetic algorithm" accepted by all in the evolutionary-computation community that differentiates GAs from other evolutionary computation methods. However, it can be said that most methods called "GAs" have at least the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring. Inversion - Holland's fourth element of GAs - is rarely used in today's implementations, and its advantages, if any, are not well established.

The chromosomes in a GA population typically take the form of bit strings. Each locus in the chromosome has two possible alleles: 0 and 1. Each chromosome can be thought of as a point in the search space of candidate solutions. The GA processes populations of chromosomes, successively replacing one such population with another. The GA most often requires a fitness function that assigns a score (fitness) to each chromosome in the current population. The fitness of a chromosome depends on how well that chromosome solves the problem at hand.

One common application of GAs is function optimization, where the goal is to find a set of parameter values that maximize, say, a complex multiparameter function. As a simple example, one might want to maximize the real-valued one-dimensional function

$$f(y) = y + |\sin(32y)|,\ 0 \leq y \leq \pi \tag{7.1}$$

[19]. Here the candidate solutions are values of *y*, which can be encoded as bit strings representing real numbers. The fitness calculation translates a given bit string *x* into a real number *y* and then evaluates the function at that value. The fitness of a string is the function value at that point.

As a non-numerical example, consider the problem of finding a sequence of 50 amino acids that will fold to a desired three-dimensional protein structure. A GA could be applied to this problem by searching a population of candidate solutions, each encoded as a 50-letter string such as

IHCCVASASDMIKPVFTVASYLKNWTKAKGPNFEICISGRTPYWDNFPGI,

where each letter represents one of 20 possible amino acids. One way to define the fitness of a candidate sequence is as the negative of the potential energy of the sequence with respect to the desired structure. The potential energy is a measure of how much physical resistance the sequence would put up if forced to be folded into the desired structure—the lower the potential energy, the higher the fitness. Of course one would not want to physically force every sequence in the population into the desired structure and measure its resistance—this would be very difficult, if not impossible. Instead, given a sequence and a desired structure (and knowing some of the relevant biophysics), one can estimate the potential energy by calculating some of the forces acting on each amino acid, so the whole fitness calculation can be done computationally.

These examples show two different contexts in which candidate solutions to a problem are encoded as abstract chromosomes encoded as strings of symbols, with fitness functions

defined on the resulting space of strings. A genetic algorithm is a method for searching such fitness landscapes for highly fit strings.

7.5 GA operators

The simplest form of genetic algorithm involves three types of operators: selection, crossover (single point), and mutation.

Selection This operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce.

Crossover This operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100100. The crossover operator roughly mimics biological recombination between two single-chromosome (haploid) organisms.

Mutation This operator randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).

Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows:

1. Start with a randomly generated population of n 7-bit chromosomes (candidate solutions to a problem).

2. Calculate the fitness $f(x)$ of each chromosome x in the population.

3. Repeat the following steps until n offspring have been created:

    a. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.

    b. With probability p_c (the "crossover probability" or "crossover rate"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents. (Note that here the crossover rate is defined to be the probability that two parents will cross over in a single point. There are also "multi-point crossover" versions of the GA in which the crossover rate for a pair of parents is the number of points at which a crossover takes place.)

    c. Mutate the two offspring at each locus with probability p_m (the mutation probability or mutation rate), and place the resulting chromosomes in the new population.

    If n is odd, one new population member can be discarded at random.

4. Replace the current population with the new population.

5. Go to step 2.

Each iteration of this process is called a *generation.* A GA is typically iterated for anywhere from 50 to 500 or more generations. The entire set of generations is called a *run.* At the end of a run there are often one or more highly fit chromosomes in the population. Since randomness plays a large role in each run, two runs with different random-number seeds will generally produce different detailed behaviors. GA researchers often report statistics (such as the best fitness found in a run and the generation at which the individual with that best fitness was discovered) averaged over many different runs of the GA on the same problem.

The simple procedure just described is the basis for most applications of GAs. There are a number of details to fill in, such as the size of the population and the probabilities of crossover and mutation, and the success of the algorithm often depends greatly on these details. There are also more complicated versions of GAs (e.g., GAs that work on representations other than strings or GAs that have different types of crossover and mutation operators). Many examples will be given in later chapters.

As a more detailed example of a simple GA, suppose that l (string length) is 8, that $f(X)$ is equal to the number of ones in bit string x (an extremely simple fitness function, used here only for illustrative purposes), that n (the population size) is 4, that $p_c = 0.7$, and that $p_m = 0.001$. (Like the fitness function, these values of l and n were chosen for simplicity. More typical values of l and n are in the range 50-1000. The values given for p_c and p_m are fairly typical.)

The initial (randomly generated) population might look like this:

Chromosome label	**Chromosome string**	**Fitness**
A	00000110	2
B	11101110	6
C	00100000	1
D	00110100	3

A common selection method in GAs is *fitness-proportionate selection,* in which the number of times an individual is expected to reproduce is equal to its fitness divided by the average of fitnesses in the population. (This is equivalent to what biologists call "viability selection.")

A simple method of implementing fitness-proportionate selection is "roulette-wheel sampling" [20] which is conceptually equivalent to giving each individual a slice of a circular roulette wheel equal in area to the individual's fitness. The roulette wheel is spun, the ball comes to rest on one wedge-shaped slice, and the corresponding individual is selected. In the $n = 4$ example above, the roulette wheel would be spun four times; the first two spins might choose chromosomes B and D to be parents, and the second two spins might choose chromosomes B and C to be parents. (The fact that A might not be selected is just the luck of

the draw. If the roulette wheel were spun many times, the average results would be closer to the expected values.)

Once a pair of parents is selected, with probability p_c they cross over to form two offspring. If they do not cross over, then the offspring are exact copies of each parent. Suppose, in the example above, that parents B and D cross over after the first bit position to form offspring E = 10110100 and F = 01101110, and parents B and C do not cross over, instead forming offspring that are exact copies of B and C. Next, each offspring is subject to mutation at each locus with probability p_m. For example, suppose offspring E is mutated at the sixth locus to form E' = 10110000, offspring F and C are not mutated at all, and offspring B is mutated at the first locus to form B' = 01101110. The new population will be the following:

Chromosome label	Chromosome string	Fitness
E’	10110000	3
F	01101110	5
C	00100000	1
B'	01101110	5

Note that, in the new population, although the best string (the one with fitness 6) was lost, the average fitness rose from 12/4 to 14/4. Iterating this procedure will eventually result in a string with all ones.

The proposed methods and algorithms for model construction are explained in the next chapter.

8 MODEL CONSTRUCTION

8.1 Description of the tasks and algorithms

The main goal of this thesis is to design a method of measurements and finding of appropriate model of a static physical field based on small number of samples which can be later extended and used in a sensor network. For practical verifications a system consisting of an autonomous robot, a camera and image-recognition algorithm and a PC running an application with user interface for data calculation and transmission has been implemented.

The model-searching method presented here is based on evaluation of the least-squares error, genetic algorithm and the predictive variance explained in the following chapters. The novel approach and the main contribution of this work lie in the use of variance and predictive variance for finding accurate model with a small number of measurements and use of genetic algorithms for optimization of the nonlinear parameters of basis functions [17]. The variance is used here as the stopping criterion for adding data points or measurements necessary to construct a reliable model.

A vast body of knowledge exists in the field of adaptive control, estimation theory and statistics. Known consistency theorems, for example in [9, 10, 14] ensure us that more data will asymptotically force convergence in some sense of the estimated parameters to the parameters that generated the data.

However, for real life engineering and machine decision problems decisions based on small samples are needed. The small sample problem is not solved in the engineering and artificial intelligence literature. This problem is also strongly application domain dependent. The proposed solution should contribute to the solution of these serious practical challenges.

8.1.1 Model used for simulations

The main application task is to measure a given static physical field and provide a reliable mathematical model of this field in minimum steps [11, 12].

The modeling part of algorithm has several steps and runs iteratively. Model iterations start with several initializations. These initializations are provided by the knowledge based system or by an expert user.

For this work a static field of light will be assumed.

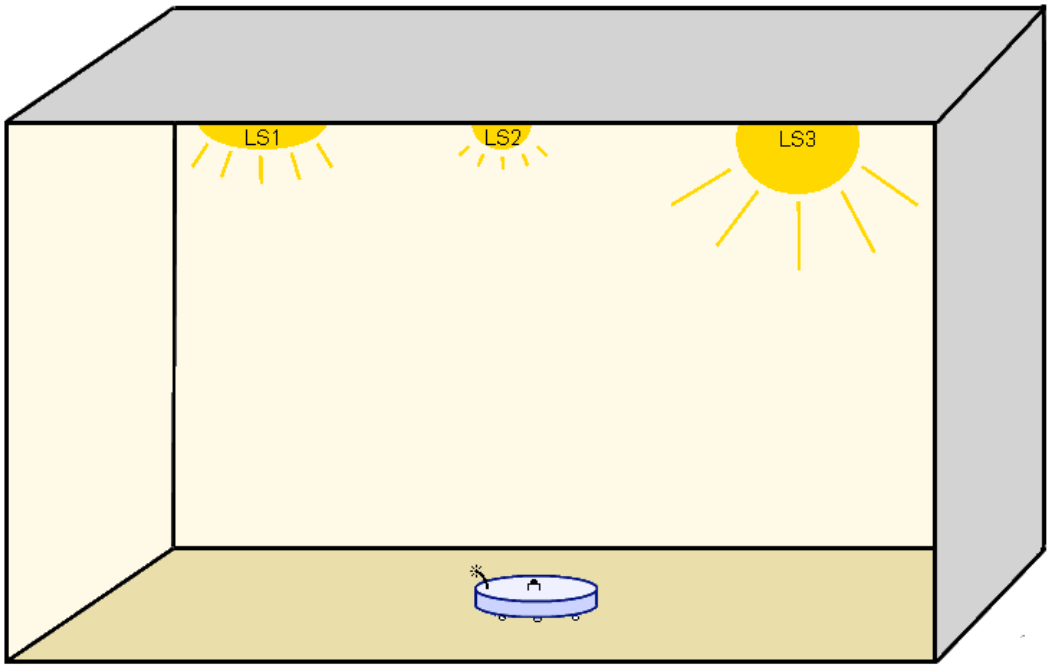


Figure 8.1 An example of the static physical field – the spatial luminous flux model has to be found

In the Fig. 8.1 a static physical light field is shown. An autonomous robot is equipped with a light sensor for luminous flux measurements. The robot can move independently in this area following the commands received wirelessly from the PC, which is computing the optimum measurement points. The basic condition and limitation for the number of measurements to be made is the fact that as few measurements as needed for providing appropriate (i.e. non-catastrophic) model can only be taken. It is desirable (e.g. due to remaining energy of the robot, or a hazardous area restrictions) to stop the process of model improvements by adding data points as soon as a criterion will tell us to do so. The criterion used in this work is the variance of the model and model error as well.

The light field consists of three light sources marked *LS1*, *LS2* and *LS3*. Each of these sources is described by distinct parameters of the luminous flux [lm], location of the center [mm] and size of width of the radiating light source [mm].

To mathematically approximate luminous flux as a function of the location measured by the robot the radial basis functions described in Chapter 3 were selected.

For the example in Fig.8.1 three radial basis functions are used as there are three light sources.

Each of the sources can be modeled as a radial basis function with a specific luminous flux, location of the center of the source and its width:

$$y(w,x,s) = w.e^{-\frac{(x-c)^2}{s}} \tag{8.1}$$

The parameter w is the lighting power, the parameter c is the location of the source's center and width s specifies the radiation width of the source.

The sum of the three radial basis functions can be used to model this static physical field.

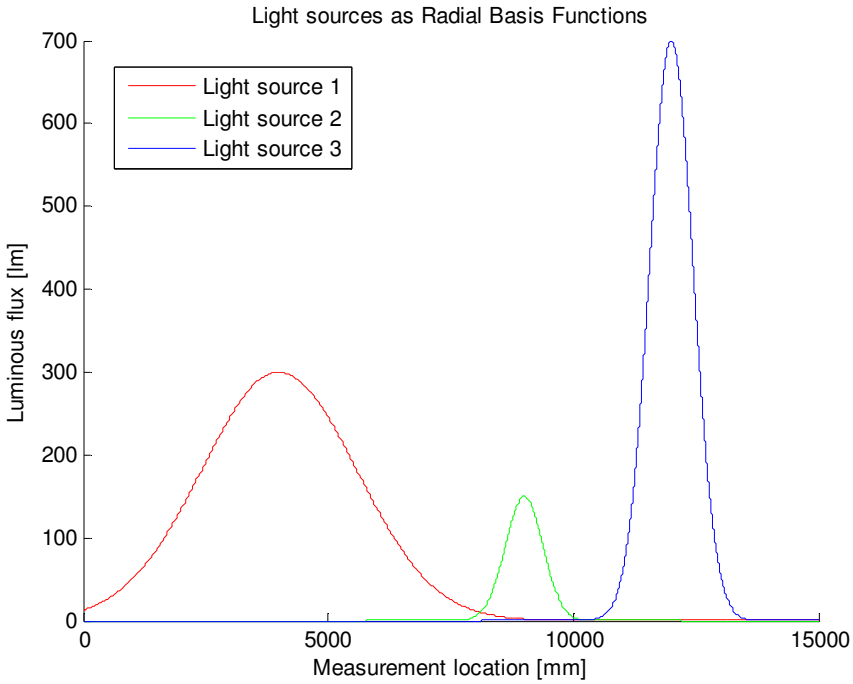


Figure 8.2 Sources of light as radial basis functions

The sources of light shown as radial basis functions in Fig. 8.2 can be modeled as the following model, which is the sum of all three light sources (i.e. basis functions):

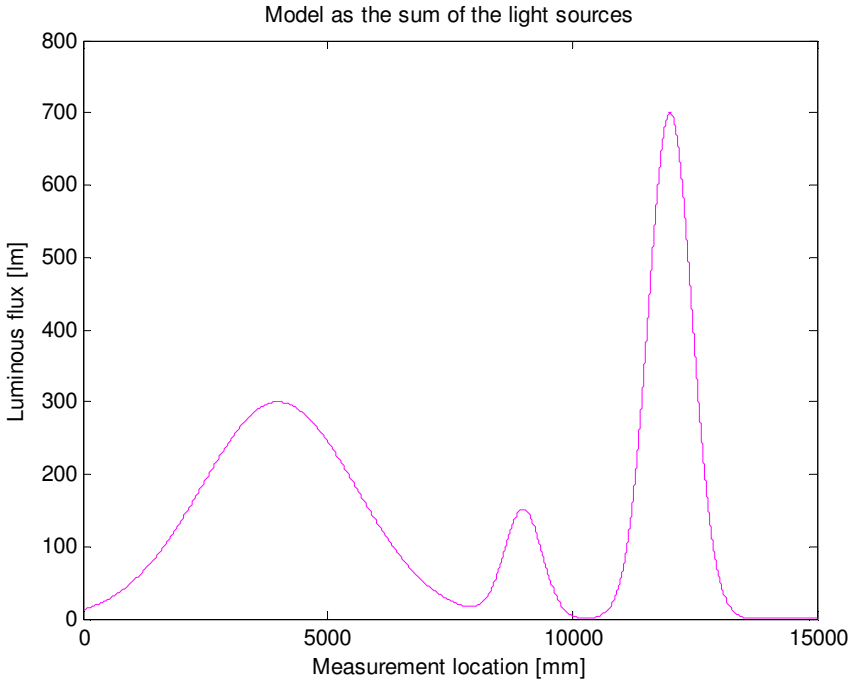


Figure 8.3 Model as sum of the three light sources

Before the proposed algorithm is carried out a knowledge-base system or an expert user enters the following parameters based on his observation:

1. Type of basis functions – for the luminous flux spatial model Radial Basis Functions were selected in this work.
2. Number of basis functions – this parameter corresponds with an estimated number of light sources. In case of insufficient number of basis functions this can be later identified throughout the run of the algorithm.
3. Determine the initial set of *n* measurements (e.g. $n = 5$, additional measurements are processed recursively or as a batch).
4. Provide the initial set of non-linear coefficients (e.g. locations of centers and the size of width of radial basis functions).

After the initialization values are provided the iterative part of the algorithm is executed.

1. The first iteration starts with analytical solving for the linear coefficients (e.g. weights in case of radial basis functions).
2. Genetic algorithm is used to optimize the non-linear coefficients (e.g. centers and widths) using the last update of the weights. This and the previous step are repeated until the error is smaller than the pre-set value or this setup is rejected (because the error does not decrease sufficiently).
3. After the least squares model is acceptable the variance is checked for excessive values. As is illustrated in Fig. 8.4a and Fig. 8.4b large variance admits very different models (the term *catastrophic* models is used in this work). If model variance is acceptable the whole procedure is done. For unacceptable variance a new measurement location is provided and the iteration process continues.

The above iteration procedure involves repeated use of optimization procedures and linear system equation solver. The linear equations are solved as a part of linear regression analysis. The optimization procedures are needed for estimating non-linear model parameters and also the variance analysis may deal with multiple minima/maxima problems. For optimization we have utilized standard genetic algorithms as described in [16], and using MATLAB. The GA method is based on three operations, namely fitness selection, crossover and mutation.

8.1.2 Models and variance

The mathematical core of the field model is assumed in the form:

$$y(x_i) = \sum_{i=1}^{m} w_i \phi(x_i) + e(i) \tag{8.2}$$

where *y* is the observed variable (luminous flux), *w* is the vector of unknown parameters and ϕ are functions from a library of known functions that may depend on other known and unknown parameters, *e (i)* is noise with variance σ^2. The model (8.2) is indexed by the

variable *i* which often denotes either the measurement number or time. The ϕ's are sometimes called regression variables, sometimes basis functions, *m* is the number of these functions, *n* denotes the number of measurement points.

In the typical example for static systems ϕ represents radial basis function, polynomials, trigonometric functions, hyperbolic tangents, for example:

$$e^{-\frac{(x-c)^2}{s}}, x^r, \sin(x), \frac{1}{(x+k)^r}, \frac{1}{1+e^{-x}}, \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

In the typical dynamic system $\phi's$ represent system state variable, for example position *x*, velocity *v*, system input *u (2), (3).* For similar examples see [2] and [3].

$$x_{k+1} = x_k + t.v_k + \frac{t^2}{2}(u_k + noise) \tag{8.3}$$

$$v_{k+1} = v_k + t(u_k + noise) \tag{8.4}$$

Dynamic models are relevant for tasks such as modeling and navigation; static models are usually used for modeling and pattern recognition.

In the simplest scenario the unknown parameters *w* are the solutions of the normal equation:

$$\hat{w} = (\Phi(n)^T \Phi(n))^{-1} \Phi(n)^T Y(n) \tag{8.5}$$

where Φ is $n \times m$ matrix with entries $\Phi(i, j) = \phi_j(x_i)$.

The model variance functions [10] as in Chapter 6 of this work, or in the neural network Bayesian context [3], can be written as

$$\text{var}(y(x)) = \sigma^2 \phi^T(x)(\Phi^T \Phi)^{-1} \phi(x) \tag{8.6}$$

Here both the variance function and the model error function are used to decide whether more measurements are needed or the model is complete. The usual stopping criteria do not use the variance function.

8.2 Use of variance as the model criterion (stopping criterion)

Assume the following situation:

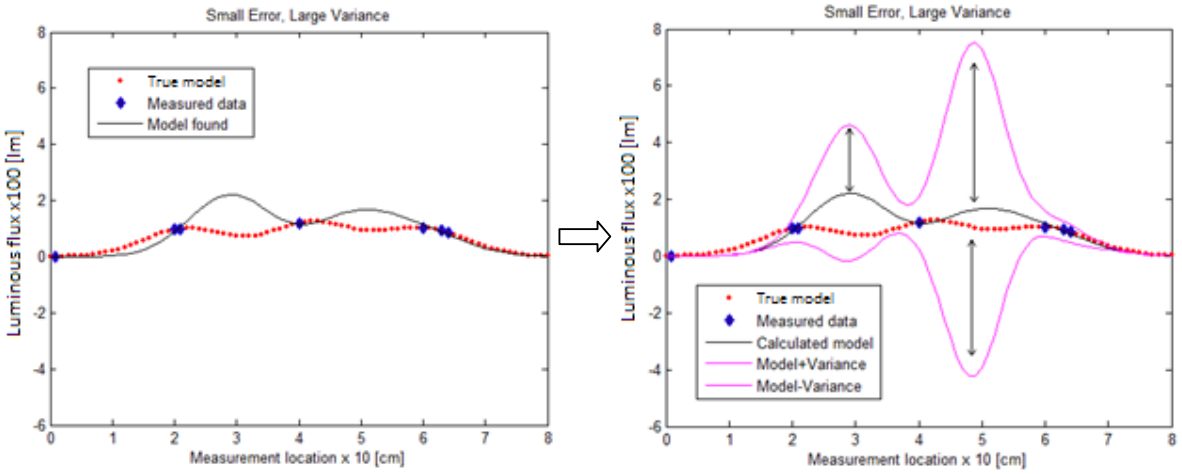


Figure 8.4a Model fit without variance

Figure 8.4b Model fit with variance

Fig.8.4a shows a typical situation where a system can be conveniently modeled with three radial basis functions. The true model has centers in points $c_1 = 2$, $c_2 = 4$, $c_3 = 6$. The approximation of the true model has to be found. With seven initial measurement points provided the model is found. Least squares method is used to find the best fit to this data (model found – black line). This fit is constructed using two basis functions with centers $c_{1'} = 3$ and $c_{2'} = 5$.

Although the model's error is minimal in the least squares sense the limitation imposed by the small number of samples results in missing the critical measurement points might result in what can be called as "*catastrophic model*".

Definition 8.1: A *catastrophic* model is the opposite of an *admissible* model. The admissible model is close to the true model in a sense that minima and maxima correspond (are not significantly misplaced). This means that conclusions drawn from a model will not be incorrect for a given application. This definition depends on an application and for a specific application this definition can be made more precise.

For an illustration of this concept, please see Fig. 8.4a and Fig. 8.4b and also Fig. 6.1.

If an additional criterion was added to the least squares minimization it could provide improvement of the model. The proposed criterion is the variance of the model. Fig. 8.4a shows the model found without its variance, while Fig. 8.4b shows the model with the variance. Adding a data point and making a new measurement will decrease the variance as follows from the consistency theorems.

8.3 Predictive variance for new measurement point determination

When a model is found based on the current data set with the least squares minimization the variance for the model is calculated. The variance (8.6) is a function of basis functions and their parameters and the location of measurement points, therefore additional data point will lead to decrease of variance. With this new data point it will be possible to either confirm the fit found in previous run of the algorithm, or the new measurement will increase the error of the model significantly in case the previous fit was not corresponding to the true nature of the modeled field.

8.3.1 Location of the new measurement point

The location of the new measurement point calculation is done in the following manner:

The user has provided 5 initial data points:

$$\begin{aligned}[x_1, y_1] &= [10,\ 1.21] \\ [x_2, y_2] &= [34,\ 1.17] \\ [x_3, y_3] &= [58,\ 4.02] \\ [x_4, y_4] &= [79,\ 3.19] \\ [x_5, y_5] &= [97,\ \ 5.9]\end{aligned} \tag{8.7}$$

With these data the initial model fit (see Fig. 8.5) has been found. The procedure of how this data fit was obtained is explained in details in the following chapters, esp. Chapter 8.5.

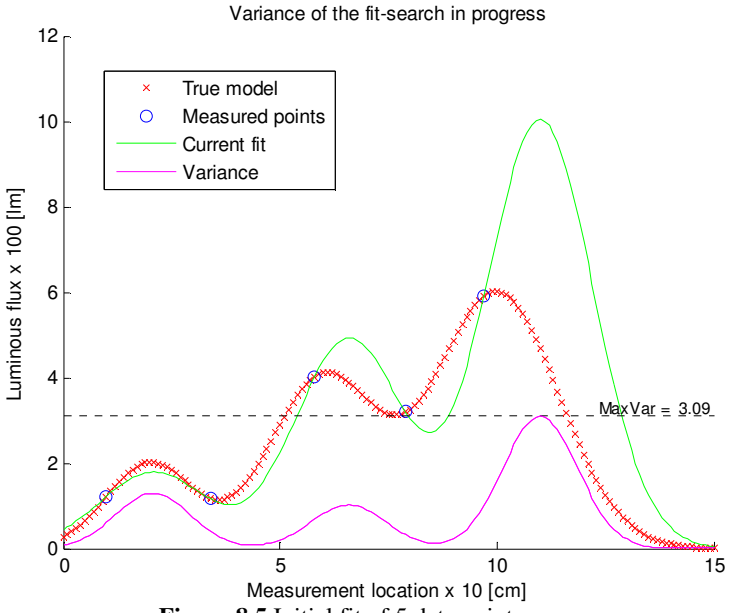


Figure 8.5 Initial fit of 5 data points

As can be seen the high variance points to a need of additional measurements (and the difference between *true* model – normally unknown, and the fit found confirms this). The data

point can be any point in the range from 0 – 150 cm in this particular scenario. The criterion for the selection of the next measurement point is:

$$xAdd \leftarrow \arg\ Min\left(Max\left(var(x)\right)\right) \tag{8.8}$$

which means that a point $x \in (x_{min}, x_{max})$ that guarantees the maximum minimization of the variance in the next measurement step will be selected. The best way how to find this point is to run a calculation of the variance for the current model with hypothetical point *x* to be added. The point which results in the variance with the lowest maximum is the point that the system will add to its current dataset, see the Fig. 8.6.

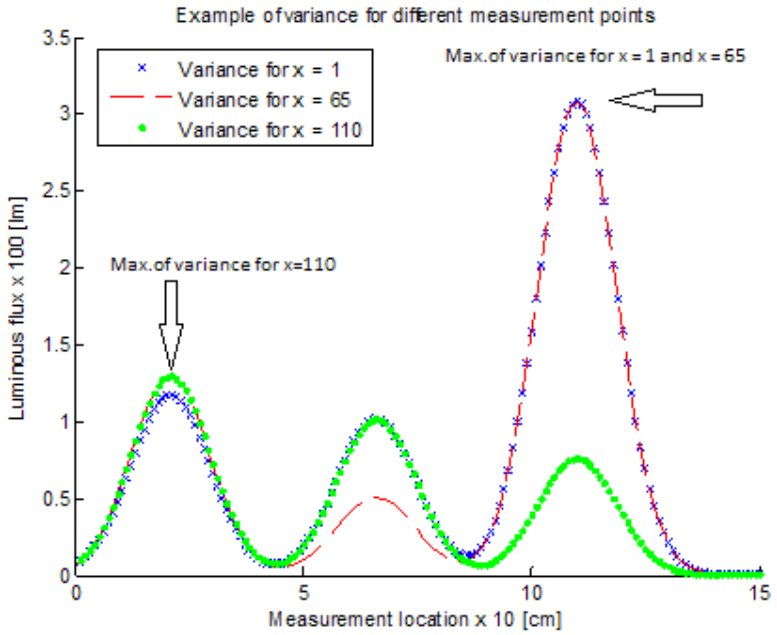


Figure 8.6 Variance for three different points

Working with the model from Fig.8.5 the data (8.7) that were used for its construction were extended with three additional data points, however one point at a time only. Fig. 8.6 shows three variances – each one for the initial data (8.7) extended with one of the "candidate" data points. At this time three points were chosen randomly:

$$x_{6'} = 1,\ x_{6''} = 65,\ x_{6'''} = 110$$

The *y*-value of the point is unknown at this moment. The variance is calculated separately for each of these points and shown in Fig. 8.6. It can be seen from this figure that the maximum of variance for $x_{6'''} = 110$ is much lower than the maximums for $x_{6'} = 1$ and $x_{6''} = 65$. Therefore if one of the three candidate data points would have to be selected as the next measurement point, it would be $x_{6'''} = 110$. However, in the algorithm described in the following chapters much greater number of candidate points is selected, in this case in the measurement search space <0…150> it would be 151 points, starting form 0 and spanning across the whole search space.

Fig. 8.7 shows in one graph all the variances found and the maximum of each variance. The point to be added to the measured that will result in the biggest variance decrease is therefore x_6= 110.

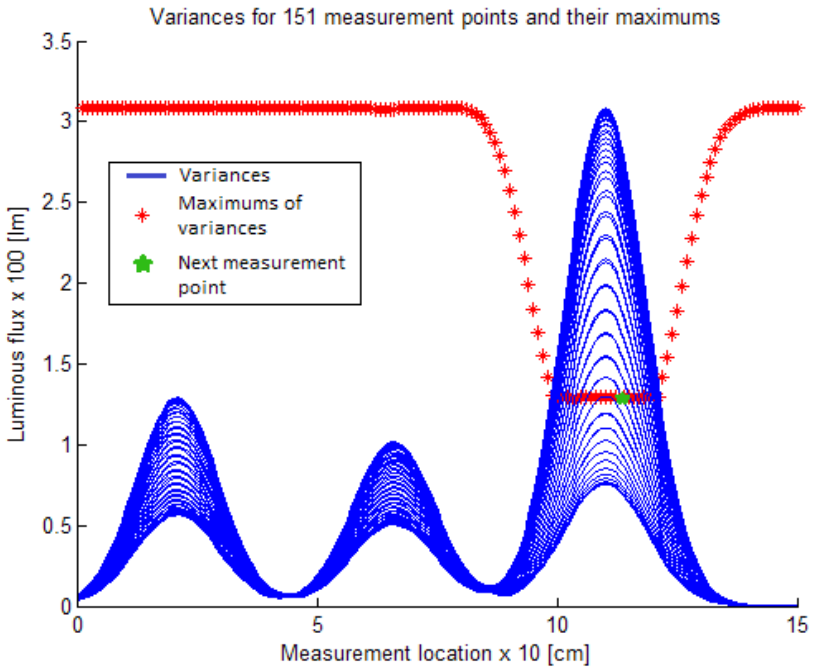


Figure 8.7 Variances for all possible measurement points and their maximums

8.4 Basis functions' parameters optimization with GA

After a measurement point has been added to the current data set and new measurement was taken, the least-squares error of the fit as in Fig. 8.5 will most probably change. If the error of the model will grow above the allowed tolerance, it will be necessary to find new weights, centers and widths of the fit so that the error will decrease again.

Since the weight parameters are "linear", they can be calculated in analytical way with the use of (8.5). However, this approach is not possible for finding the centers and widths, which are considered non-linear parameters as in (8.1). A different method has to be used in this case. The method chosen for this work utilizes genetic algorithms. Advantage of this method is fast searching through the measurement space resulting in desired values of centers and weights that minimize the model's error.

8.5 Flowchart for searching the model

In this chapter complete algorithm for searching the fits is presented and explained in greater details.

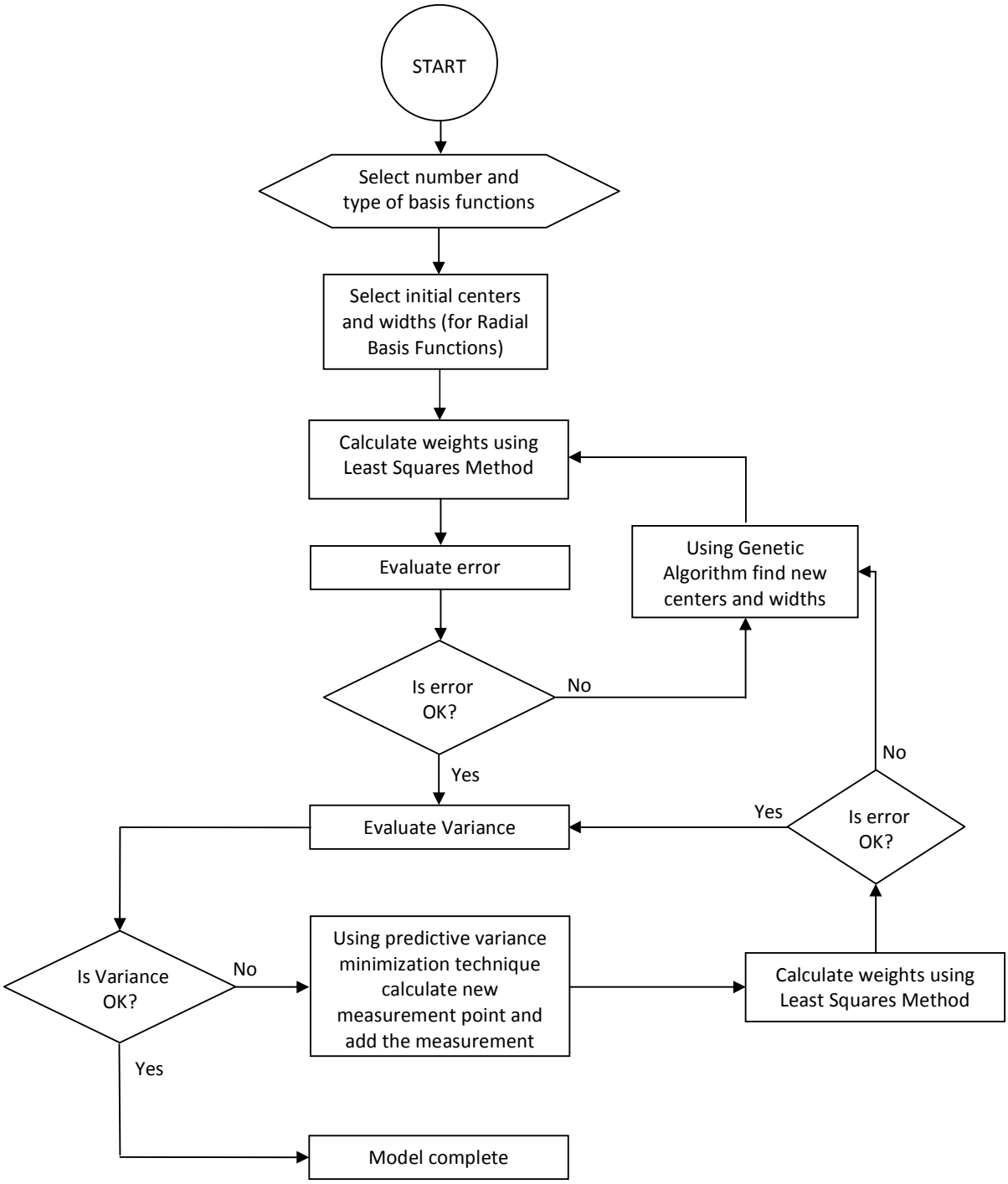


Figure 8.8 Flowchart for searching the model

The fit-searching procedure consists of the following steps as listed in the above flow diagram:

1) Select number and type of basis functions

Based on the operator experience and knowledge base the number and types of basis functions are selected. In this thesis it is assumed that the physical static field to be measured is generated by multiple light sources with unknown luminous flux and estimated locations of centers and widths of the light emission. Summing the appropriate Radial Basis Functions (RBFs) allows approximation of vast amount of functions. However, the closer the match between the type of basis functions selected and the nature of the physical process that is being identified or approximated, the more efficient the approximation is.

In the chapter 8.8.1 an example showing wrong initialization of number and type of basis functions is shown and a criterion for their identification is presented as well.

2) Select initial centers and widths (for Radial Basis Functions)

As stated above the model used in this thesis consists of Radial Basis Functions (RBF). Each radial basis function consists of three parameters, two of which are non-linear and one is linear:

$$y(w, x, s) = w.e^{-\frac{(x-c)^2}{s}} \tag{8.9}$$

where w……weight (amplitude), i.e. linear parameter
c……..center, i.e. non-linear parameter
s……..width, i.e. non-linear parameter

The initial centers and widths of the radial basis functions of the system to be approximated are estimated by the user. They will be later readjusted in the later run of the complete algorithm.

3) Calculate weights using Least Squares Method

Using the least-squares method algorithm calculate the weights for the current centers and widths to obtain the weights that guarantee minimum error:

$$\widehat{\boldsymbol{w}} = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\boldsymbol{Y} \tag{8.10}$$

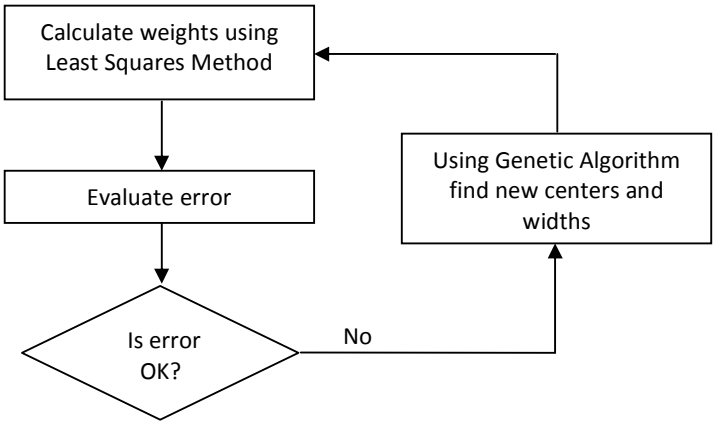
Where

$\widehat{\boldsymbol{w}}$ ……vector of weights for number of basis functions

Φ…….matrix of values of basis functions in the measurement points

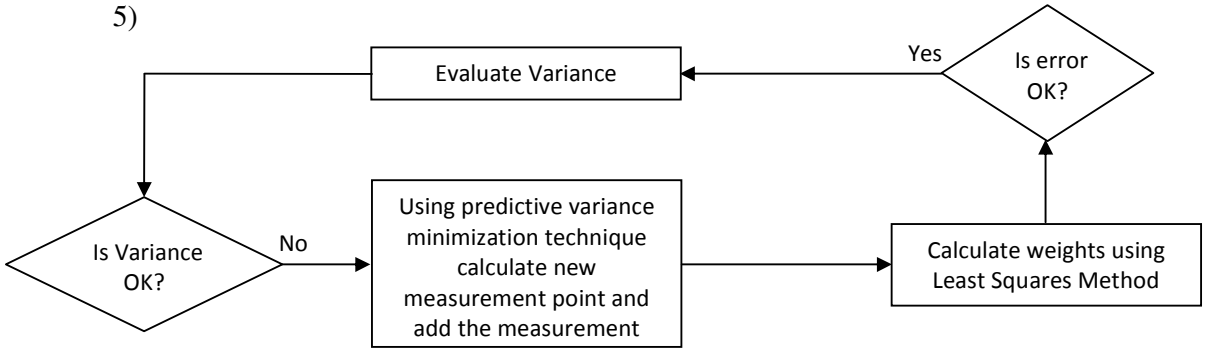
Y……..vector of measured values

4)



Previous steps resulted in the first model; therefore its error can now be evaluated. If the error exceeds the allowed tolerance, use genetic algorithm to search for new centers and widths of the basis functions. Repeat until the error of the model is smaller than the tolerance.

5)



If the error in the previous step passed the criteria evaluate variance of the current model.

If the variance exceeds the allowed variance for the model obtained in previous steps, additional data point has to be added. This data point will be calculated as the point where the variance will be minimized the most when this data point is added to the current data set. If the error of the model has not exceeded the tolerance ("Is Error OK?" decision block), that means current model fits the true model well, variance is re-evaluated, if the error has risen, the algorithm will follow with update of weights, centers and widths.

6) Steps 3) - 5) are repeated as long as both error and variance values for the current model are within specified tolerances.

8.6 Flowchart for searching a model with admissible variance and error

This chapter described the process of searching maximum admissible variance for various models and numbers of radial basis functions. In this scenario knowledge of the *true* model is assumed and used in the calculations. An operator runs the following algorithm multiple times and evaluates the variance and error criteria until the group of fits found contains no catastrophic fits. The variance obtained is then stored into the knowledge base and an approximation function for calculating variance limit is obtained.

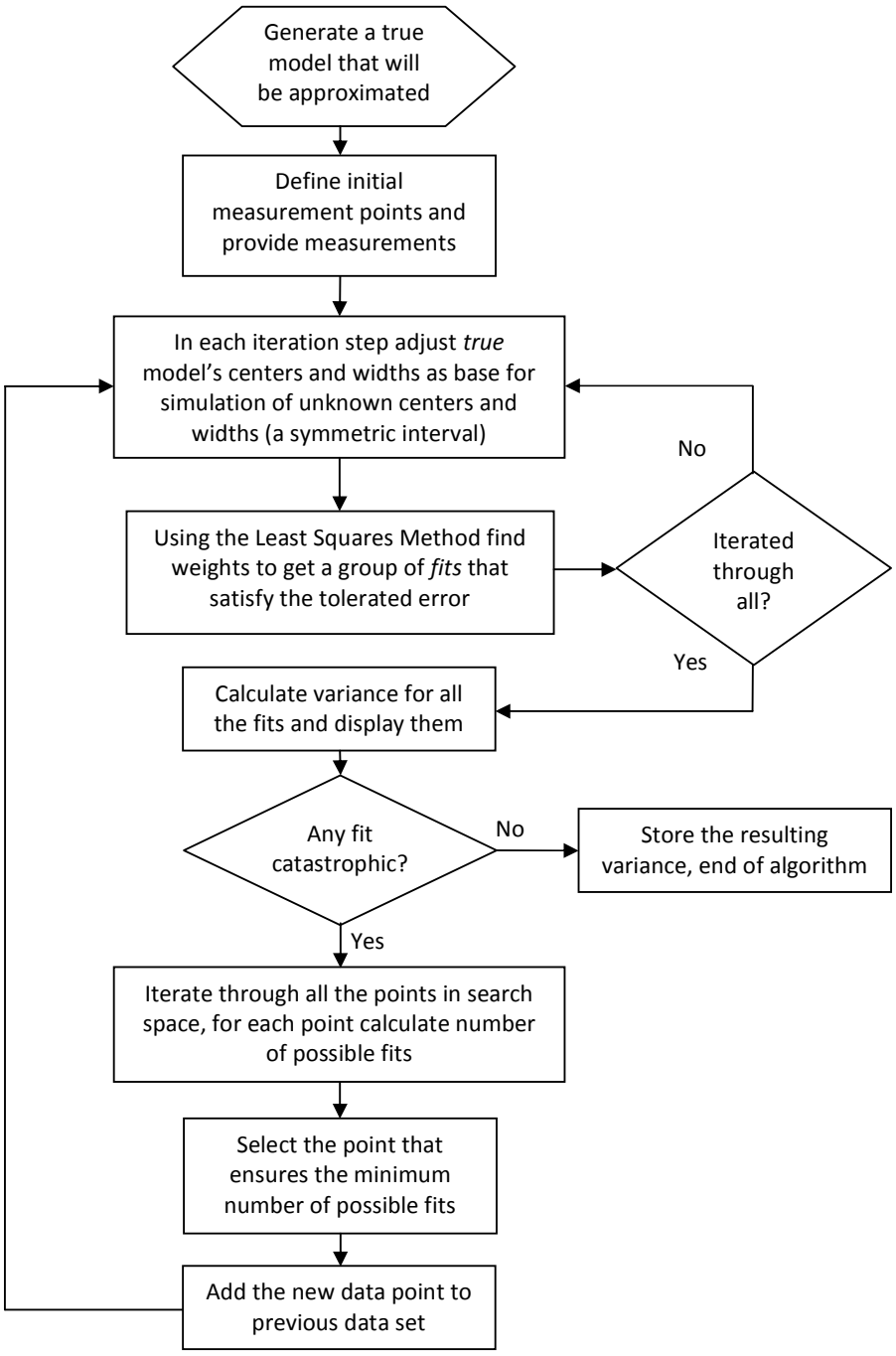
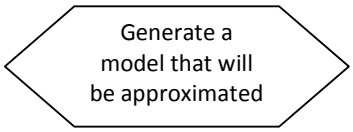
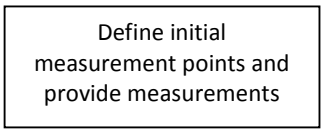


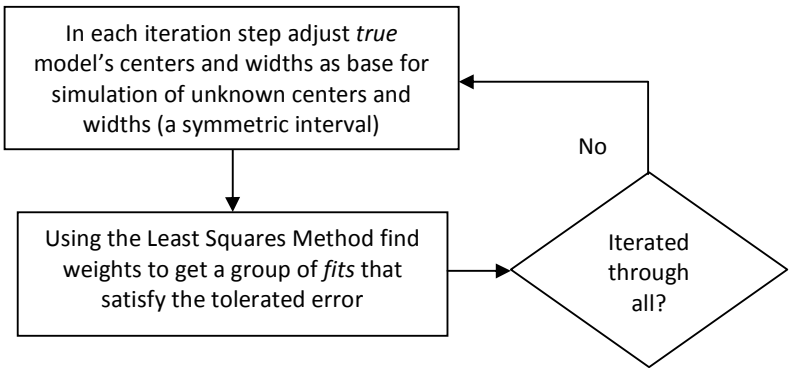
Figure 8.9 Finding admissible variance

1) 

Select *true* model of the class of models similar to the model being sought (2 and 3 radial basis functions are used in this work).

2) 

Select small number of initial measurement points as if the true model was unknown and provides initial measurement points and measurements based on the selection.

3) 

Assuming that the centers and widths of the good fit for the static field we are trying to model are unknown, use the centers and widths of the *true* model with a symmetric interval around the centers and widths to simulate the uncertainty of the unknown fits.

For example, in case of the *true* model consisting of:

$$y_1 = 77.e^{-\frac{(x-3.5)^2}{2}}$$
$$y_2 = 40.e^{-\frac{(x-6.5)^2}{2.5}}$$
$$y_3 = 50.e^{-\frac{(x-10)^2}{3.5}}$$

Where the *true* model *y* is:

$$y = y_1 + y_2 + y_3$$

and vector of centers of the model:

$$\boldsymbol{c} = [c_1, c_2, c_3] = [3.5, 6.5, 10]$$

and vector of widths of the model:

$$\boldsymbol{s} = [s_1, s_2, s_3] = [2, 2.5, 3.5]$$

To find all the fits with an error lower than the tolerance value it is necessary to iterate through all the possible combinations of centers and widths. However, as some knowledge of the static field to be measured is assumed, we can limit the search-space around the *true* centers and widths to some reasonable value, e.g.

For centers

$$\boldsymbol{c} = [3.5, 6.5, 10]$$

the search-space will be:

$$\boldsymbol{c} = [3.5 \pm 1.5, 6.5 \pm 1.5, 10 \pm 1.5]$$

and for widths

$$\boldsymbol{s} = [2, 2.5, 3.5]$$

The search space will be:

$$\boldsymbol{s} = [2 \pm 1, 2.5 \pm 1, 3.5 \pm 1]$$

The resolution for the search-space depends on the time capacity and the computing power.

For each combination of centers and widths the Least Squares Method is used to calculate the weights of the fit.

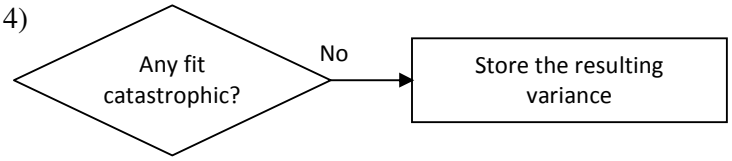
The vector of the weights being the only linear parameters of the model denoted as:

$$\boldsymbol{w} = [w_1, w_2, w_3]$$

is entered into the current fit.

The error of the model defined as the sum of squares of error in the measurement points is evaluated and all the fits with error lower than the allowed maximum are stored.

4)

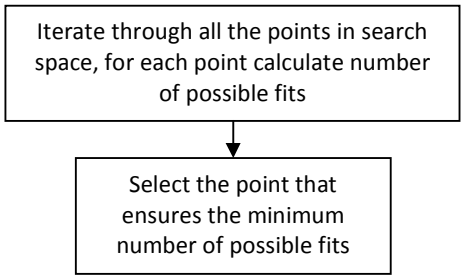


After we iterated through all the possible fit (combinations of centers and widths in the search-space) every fit has to be classified as either *catastrophic,* or *admissible*.

The meaning of the *catastrophic* fit (model) is defined in Defintion 8.1. *Non-catastrophic* model is the model which resembles the *true,* or *admissible* model in the way that the local/global extremes of the model are not placed in significantly different locations than those of the *true* model, and the centers, widths and weights of the fit are also placed close to the true model's parameters.

If no *catastrophic* fit was found in this step, the resulting variance is stored into the knowledge database and algorithm ends.

5)

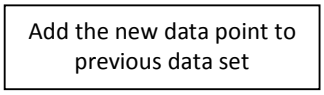


In case when the previous steps did find the catastrophic fits in the group of possible fits with error smaller than the allowed value, additional data point has to be measured.

Since possibly hundreds or thousands of fits and catastrophic fits that satisfy the allowed error may have been found, it is necessary to choose one point that will be added to current data points that reduces the number of possible fits as much as possible. This step requires as many iterations as there are possible data points in the measurement domain (search-space). For each added point the number of fits that satisfy the allowed error value is stored. Then of all this points the point that allows the fewest fits is selected as the next measurement point. See Chapter 9 for an example.

The consistency theorem ensures that the fit converges to the true model with additional data points.

6)



The data point obtained from the previous step is added to the data set, new measurement is taken and the process of finding all the fits with error smaller than the allowed value is repeated.

8.7 Implementation in MATLAB

For the simulation and verification various tools were used. The simulation algorithms were implemented in Matlab, which is a general computation/simulation environment for research and control.

In order to make the code transparent and simple a number of functions were implemented. These functions reflect the theory presented in the theoretical part of this thesis and the method proposed in the Chapter 8.

FitSearch.m

This file contains the code for running the algorithm described in the flowchart in Fig. 8.8 to find the model based on the measurement points and admissible values of error and variance.

For the simulation purposes it is assumed that the *true* model is known so at any step of the algorithm actual model can be compared to the target (true) model.

Inputs to the algorithm are:

- Vector of centers of the true model $\boldsymbol{c} = [c_1, c_2, \dots, c_n]$, where n is number of basis functions
- Vector of widths of the true model $\boldsymbol{s} = [s_1, s_2 \dots s_n]$, where n is number of basis functions
- Vector of weights of the true model $\boldsymbol{w} = [w_1, w_2 \dots w_n]$, where n is number of basis functions
- Maximum admissible variance of the model: *AllowedVar* = 0.85
- Maximum admissible error of the model: *AllowedErr* = 0.4 (application dependent)
- Vector of initial measurement points and their values provided by user: $xVals = [x_1, x_2, \dots x_n]$, $yValsGood = [y_1, y_2, \dots y_n]$ where n is number of initial measurement points
- Vectors of estimated centers and widths of the model provided by user: ***centFit*** = $[cg_1, cg_2, \dots, cg_n]$, ***widthFit*** = $[sg_1, sg_2, sg_3]$,

CatFit.m

This file contains the code for running the algorithm described in the flowchart in Fig. 8.9 to find the maximum admissible values of variance for a reliable model. Inputs and other variables are the same as in FitSearch.m, except for the value *AllowedVar* which is result of the code in this file.

Helper functions programmed in Matlab

Additional helper functions used in the FitSearch.m and CatFit.m program (only algorithm-related computation functions shown here) are:

rbf.m

```
function y=rbf(c, s, t)
```

This function returns the vector of values of the radial basis function with centers ***c*** and widths ***s*** in the points given by vector ***t***.

FindWeights.m

```
function [w1 w2 w3 guessErr] = FindWeights(xVals, yVals, centGuess,
widthGuess)
```

Inputs to this function are actual estimates of vectors of centers and widths of the model being sought, measurement points provided by user and their values.
Returned are the weights resulting from the least squares method (three weights for three basis functions) and the error of the model found in this way with respect to values of the measurement points.

Additional function created is FindWeights2BF for a different number of basis functions than three.

GetFitnessPV.m

```
function error = GetFitnessPV(coeffs, weights, xVals, yVals)
```

Since the fit-search algorithm relies on the genetic algorithm used in the search for the optimal centers and weights with minimum error an objective function had to be provided in order to assess the error of the last model resultant from the genetic algorithm.

Inputs to this function are current coefficients of centers, widths, weights, measurement points and their values.
Return value is the error in the least squares sense.

EvalError.m

```
function [ErrOK, error] = EvalError(xVals, yVals, weights, centers,
widths, allowedErr)
```

Function evaluating the error of current model given by input parameters of measurement points, measured data, centers, widths and weights.
On the exit this function sets also *ErrOK* flag which is either true or false based on comparison with *allowedErr* input value. True is returned if error of current model is than admissible error value, false otherwise.

EvalVariance.m

```
function [VarOK, Variance, MaxVar] = EvalVariance(xVals, centers,
widths, allowedVar)
```

This function calculates variance for the current model (measurement points, centers, widths) and returns vector of variance, maximum of the variance and a flag *VarOK* with the value of *true* when maximum variance is smaller than the admissible variance, false otherwise.

FindLowestVariance.m

```
function maxVarPoint = FindLowestVariance(xVals, centers, widths,
rangeLow, rangeHigh)
```

This function is used to calculate the point to be added to current measurement points that guarantees maximum minimization of the variance after this new point will be added to current model (based on measurement points *xVal,* centers and weights).

The returned value *maxVarPoint* is a point between *rangeLow* and *rangeHigh*, which is usually the measurement domain.

FindFits.m

```
function [catFitsWeights,catFitsCenters,catFitsWidths,catFitsErrors,
fitsFound] = FindFits(xVals,yVals,centers,centInt, widths, widthInt,
trueWeights, res, LSerr)
```

This function searches for all the possible fits (returned from the function as vectors *catFitsWeights, catFitsCenters, catFitsWidths*) with error smaller than input parameter *LSerr* based on current measurement points and their values (*xVals, yVals*), centers vector *centers* and widths in vector *widths*. The input variables *centInt* and *widthInt* denote the interval in which the centers and weights are searched with the resolution *res*. Since in this case all the possible combinations of vectors of centers and widths are searched within the interval given by [*centers - centInt, centers + centInt*], [*widths – widthInt, widths + widthInt*] it is necessary to restrict the search space by the resolution *res* to bring the computation time down to acceptable values.

Returned are the matrices *catFitsWeights, catFitsCenters and catFitsWidths* and the vector *catFitsErrors.* The matrices contain in each line vector holding weights, centers and widths of each fit and *catFitsError* contains values of error of each fit.

GetVariance.m

```
function S = GetVariance(xVals, centers, widths, rangeLow, rangeHigh)
```

According to the current measurement points *xVals*, centers and widths the variance is calculated and returned in 1-dimensional vector from this function.

8.8 Model searching – numerical example

In this example a model is being sought.

The centers of the true model are: $\boldsymbol{c} = [2, 6, 10]$
The widths of the true model are: $\boldsymbol{s} = [2, 3, 4]$
The weights of the true model are: $\boldsymbol{w} = [2, 4, 6]$

Initial estimates of the centers are: $\boldsymbol{centFit} = [1.7, 6.6, 10]$
Initial estimates of the widths are: $\boldsymbol{widthFit} = [3, 3, 3]$
Initial measurement points are: $\boldsymbol{x} = [1.0, 3.4, 5.8, 7.9, 9.7]$

The target error and variance to reach in the searching the model are:

$AllowedErr = 0.4$ (selected by user based on the application)
$AllowedVar = 0.85$ (See Chapter 9)

1)

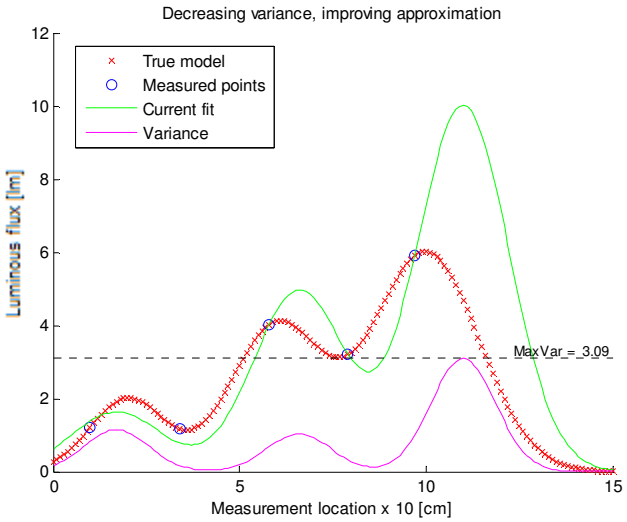


Figure 8.10 Searching model – step 1

This is the initial state of the model search after calculating the weights the analytical way in the first step.
The weights are calculated as $\boldsymbol{weightsFit} = [1.63, 4.96, 10.02]$

Centers and widths remain unchanged:

$\boldsymbol{centFit} = [1.7, 6.6, 10]$
$\boldsymbol{widthFit} = [3, 3, 3]$

Error of this fit is: $ErrorFit = 0.4233$

Since the error is greater than the allowed error, the centers and widths of the model need to be optimized.

2) The first optimization of centers and widths by genetic algorithm leads to the fit in Fig. 8.11.

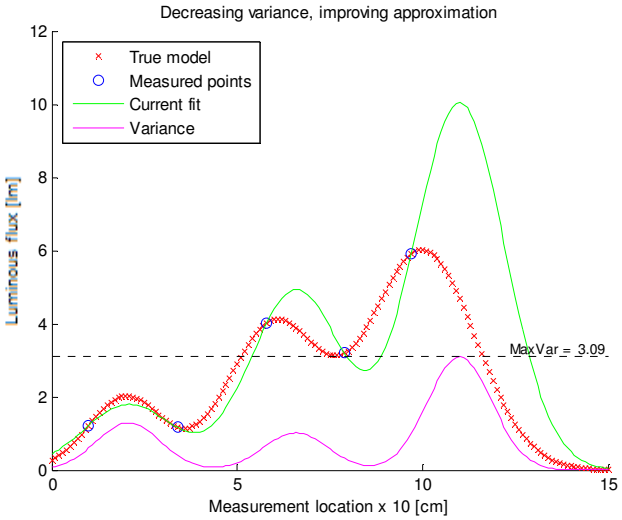


Figure 8.11 Searching model – step 2

The centers and widths were now optimized to:
$\boldsymbol{centFit} = [2.07, 6.59, 11.00]$
$\boldsymbol{widthFit} = [3.00, 3.027, 3.00]$

The weights were recalculated to
$\boldsymbol{weightsFit} = [1.79, 4.92, 10.03]$

The main improvement in this step is that the center of the first radial basis function matches much better the true model.

Error has decreased significantly to:

$$ErrorFit = 0.0176$$

which is less than the required value of 0.4. Therefore the next step is evaluation of variance and possible addition of the next measurement point.

3) Since the maximum variance of 3.09 (see Fig. 8.11) of the previous fit was greater than the allowed maximum variance of 0.85, position of additional measurement point will be calculated with predictive variance method explained in Chapter 8.3. The resulting point is at $x = 11.1$. See Fig. 8.12.

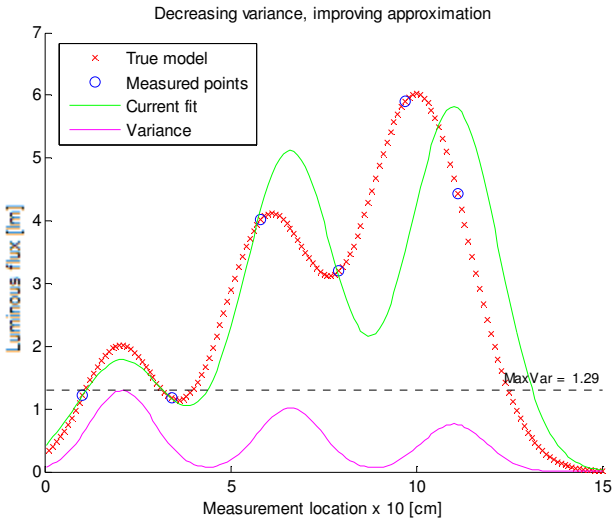


Figure 8.12 Searching model – step 3

The new measurement point is added, variance drops somewhat and weights are updated as shown in flowchart in Fig. 8.8:

$\boldsymbol{weightsFit} = [1.78, 5.11, 5.82]$

Centers and widths remain unchanged:
$\boldsymbol{centFit} = [2.07, 6.59, 11.00]$
$\boldsymbol{widthFit} = [3.00, 3.027, 3.00]$

Although the weights have dropped in this step and approximate the true model better now, the additional data point results in increase of the model's error due to misplaced centers of the radial basis function two and three.

$ErrorFit = 2.75$
$Allowed\ Error = 0.4$

With this increase of the error the centers and widths need to be optimized by genetic algorithm in the next step.

4) Optimization of the centers and widths results in better position of second and third basis function, therefore error is decreased considerably:

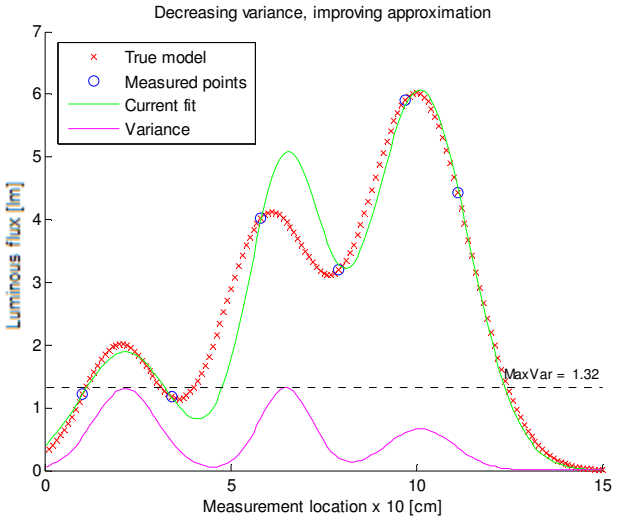


Figure 8.13 Searching model – step 4

Due to replacement of the centers variance has grown by a small amount.

$$\boldsymbol{weightsFit} = [1.89, 4.95, 6.06]$$
$$\boldsymbol{centFit} = [2.15,\ 6.48, 10.1]$$
$$\boldsymbol{widthFit} = [3.00, 2.00, 3.43]$$
$$ErrorFit = 0.16$$

Error is within the tolerance after this step, therefore since the variance is still above the allowed value, additional measurement point needs to be found.

5) Predictive variance algorithm selects point $x = 4.6$ and after adding it to current data set new variance and error are evaluated.

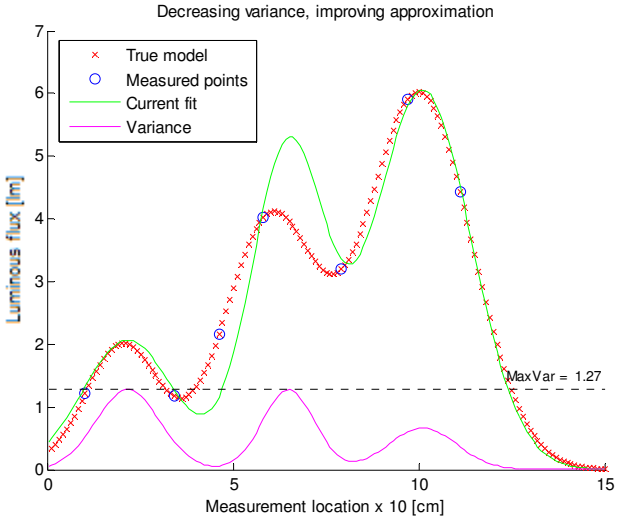


Figure 8.14 Searching model – step 5

Variance has decreased, but the new data point again increases the error of the fit:

$\boldsymbol{weightsFit} = [2.06, 5.17, 6.05]$
$\boldsymbol{centFit} = [2.15, 6.48, 10.1]$
$\boldsymbol{widthFit} = [3.00, 2.00, 3.43]$
$ErrorFit = 1.03$

With the error of the fit greater than the target value of 0.4 new centers and widths have to be found.

6)

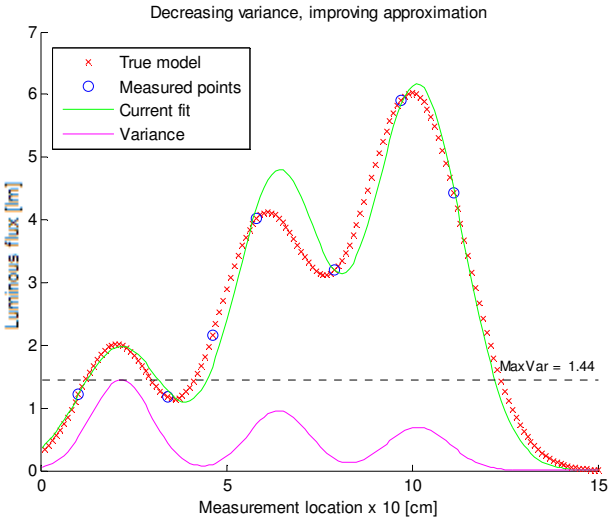


Figure 8.15 Searching model – step 6a

The first run of the genetic algorithm did not result in sufficient error decrease; therefore the algorithm will be repeated in the next step.

$\boldsymbol{weightsFit} = [1.98, 4.75, 6.15]$
$\boldsymbol{centFit} = [2.15, 6.4, 10.1]$
$\boldsymbol{widthFit} = [2.68, 2.7, 3.00]$
$ErrorFit = 0.5509$

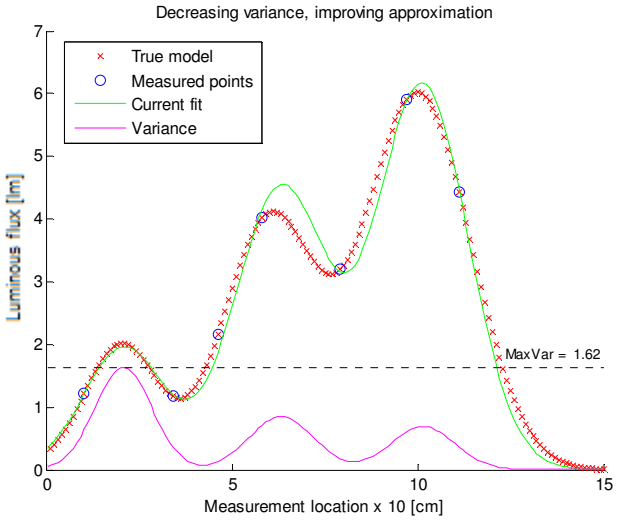


Figure 8.16 Searching model – step 6b

The second run of the genetic algorithm decreased the error below the target value of 0.4 by optimizing the centers and widths of the fit:

$\boldsymbol{weightsFit} = [1.95, 4.5, 6.13]$
$\boldsymbol{centFit} = [2.06, 6.31, 10.13]$
$\boldsymbol{widthFit} = [2.39, 3.08, 3.00]$
$ErrorFit = 0.316$

Although the error is within the limits at this point, variance is still too high. New measurement point has to be added.

7) The new data point to be added is at $x = 1.9$.

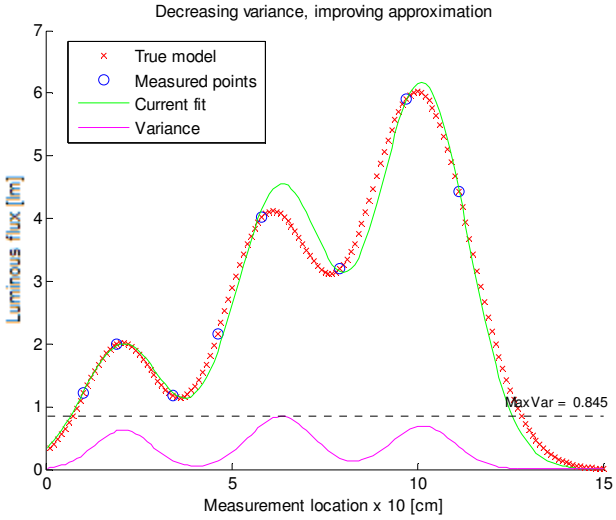


Figure 8.17 Searching model – step 7

$\boldsymbol{weightsFit} = [1.99, 4.5, 6.13]$
$\boldsymbol{centFit} = [2.06, 6.31, 10.13]$

$\boldsymbol{widthFit} = [2.39, 3.08, 3.00]$
$ErrorFit = 0.32$
$Maximum\ Variance = 0.845$ (see Fig.8.17)
In this moment both variance and error are below the required target values.
Model is done.

The comparison of the fit found with the true model shows that it was possible to successfully avoid the situation of the catastrophic model.

Since the optimization method based on genetic algorithm is a random method to some extent, multiple runs of the same algorithm can often result in even better approximation, as in Fig. 8.18.

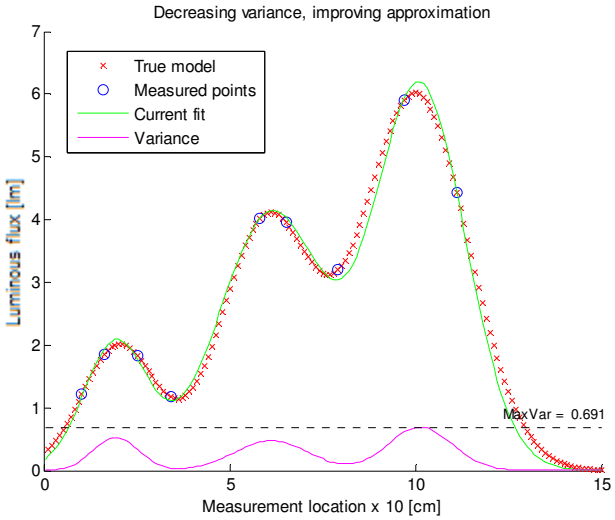


Figure 8.18 Searching model – a different result

To verify the correctness of the proposed algorithm 50 runs of the same algorithm have been executed with the following result:

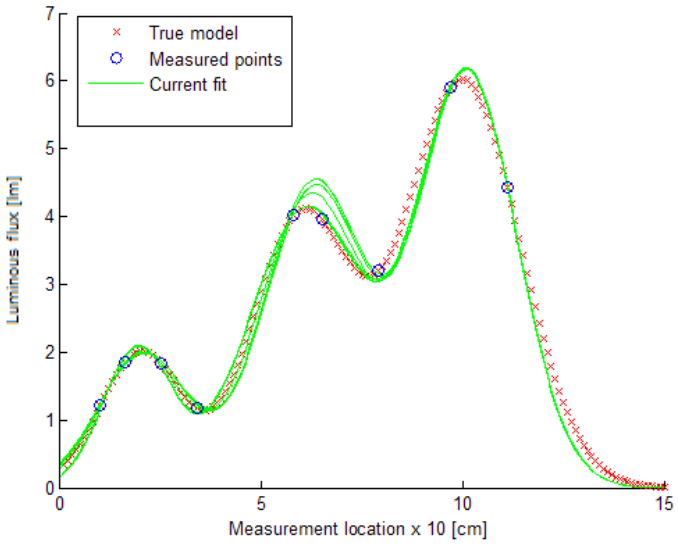


Figure 8.19 Searching model – 50 runs of the same algorithm

This proves the reliability of the algorithm. Note: the measured points in Fig.8.19 are shown only for the last of the 50 fits.

8.8.1 Management of incorrect number of basis functions

The proposed algorithm for fit searching depends on initial assumptions made by an expert user or knowledge base. One of the assumptions to be made is the number of basis functions.

In real situations, e.g. when the light sources are placed close to each other, or their widths are large and the basis functions overlap, it may be difficult to accurately estimate the number of the basis functions to be used for modeling. Therefore this situation is shown in an example.

The true model values:
The centers of the true model are: $\boldsymbol{c} = [3, 6, 9]$
The widths of the true model are: $\boldsymbol{s} = [4, 4, 4]$
The weights of the true model are: $\boldsymbol{w} = [2, 5, 4]$

Although the true model consists of three basis functions (light sources), user – based on the measurements only assumes two basis functions (light sources).

Measurement points: $xVals = [2, 4, 6, 8, 10]$
Initial estimates of the centers are: $\boldsymbol{centFit} = [6, 9]$
Initial estimates of the widths are: $\boldsymbol{widthFit} = [3, 3]$

The target error and variance to reach in the searching the model are:

$AllowedErr = 0.4$
$AllowedVar = 0.85$

1) By running the algorithm as in flowchart in Fig. 8.8 the weights of the model are calculated with least squares method:

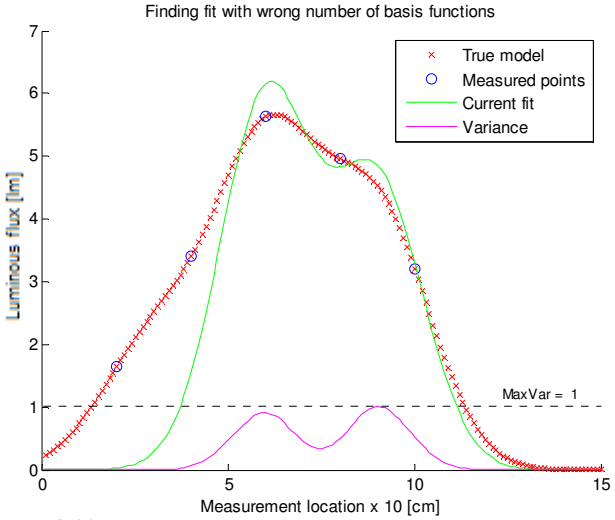


Figure 8.20 Searching model with wrong number of basis functions

$\boldsymbol{weightsFit} = [5.93, 4.56]$

Centers and widths remain unchanged, as the genetic algorithm hasn't been used so far:

$\boldsymbol{centFit} = [6, 9]$
$\boldsymbol{widthFit} = [3, 3]$

$ErrorFit = 6.31$

The error value of 6.31 is too large and needs to be optimized. Therefore the search algorithm will move on to genetic algorithm optimization of centers and widths.

2) However, since the true model consists of three basis functions and only two are assumed for the model to be found it turns out to be impossible to reduce the model's error below the required value of 0.4.

The genetic algorithm keeps running in the loop, but after 60 runs the error never falls below 0.4, see Fig. 8.21. Therefore in this moment it is recommended to either stop the searching process and re-evaluate the number of basis functions selected, or increase the error tolerance, if desired.

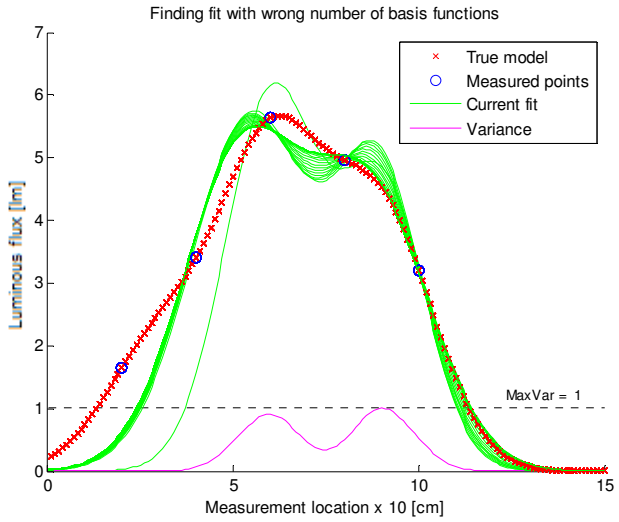


Figure 8.21 Searching model with wrong number of basis functions – 60 runs of genetic algorithm

The next chapter presents the method of searching for the value of admissible variance.

9 SEARCHING FOR ADMISSIBLE VARIANCE

The variance value of 0.85 used as stopping criterion in previous example (Chapter 8) is a value that needs to be calculated, or estimated in some way before the model searching algorithm is started. This chapter explains how this value is found.

For a certain class of models this value is very similar. Therefore it is possible to generalize this value in advance before starting searching for an unknown model assuming basic information about the model's parameters is available (types of basis functions, number of basis functions, approximate weights – luminous flux etc).

This example follows the algorithm summarized in flowchart in Fig. 8.9.

In this case a true model can randomly be selected as:

Centers of the true model: $\boldsymbol{c} = [3,\ 7, 11]$
Widths of the true model: $\boldsymbol{s} = [4, 3, 3]$
Weights of the true model: $\boldsymbol{w} = [3.5, 3.5, 6.0]$
Initial measurement points: $\boldsymbol{x} = [1.0, 3.4, 6.8, 9.0, 12.2]$
The required error value set by user is 0.4.

1) In the first step the algorithm uses the initial measurement points to find a group of fits that all satisfy the error condition, therefore looks for the fits whose error is smaller than the allowed error of 0.4. Here no genetic algorithm is used, but the algorithm iterates through all possible combinations (within some interval) of centers and widths. All the fits with error less than 0.4 are saved into an array.

Since there are only five initial points provided, the total number of fits with error less than 0.4 is 873. However, majority of these fits is "catastrophic" and the variance is high as well. The Fig. 9.1 shows the true model and three examples of catastrophic fits from the group of 873 fits found.

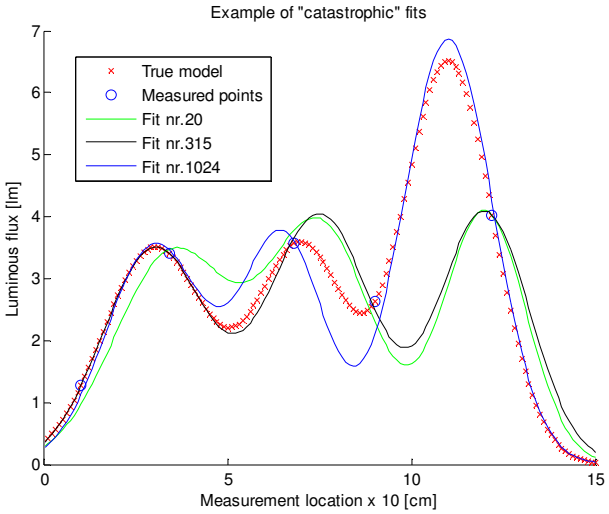


Figure 9.1 True model and three of the fits with small error

2) To reduce the variance and number of catastrophic fits a new measurement point has to be added. The algorithm for selection of the new measurement point iterates through all the measurement points in the search space from 0 to 150 with step of 2. For each of these points number of possible fits that satisfy the error condition is calculated. Therefore 75 possible measurement points are checked.

   The point $x = 8.2$ is selected as the next measurement point. Adding this point to the current set of measurement points reduces the number of possible fits with error smaller than 0.4 from 873 to 209. However, among the 209 possible fits there are still some fits which do not approximate the true model very well:

   Fits 1 and 207 are catastrophic:

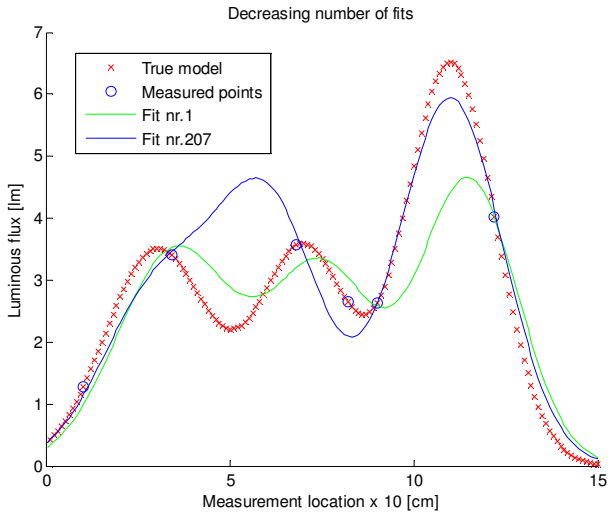


Figure 9.2 Reducing group of fits

3) Presence of catastrophic fits in the group of possible fits means another measurement point has to be selected in the same manner as in the previous step.
   This time the point $x = 5$ is selected. The number of possible fits after adding this point drops down to 101 from 209.

   Example of catastrophic fits from this group are fits 4 and 33:

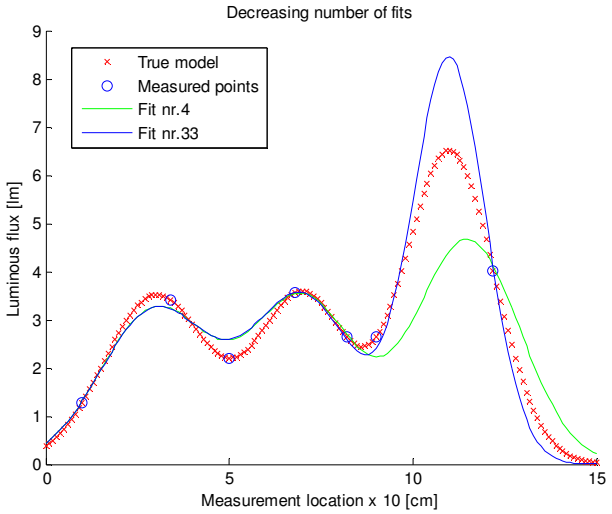


Figure 9.3 Reducing group of fits

4) The next measurement point is $x = 11$. The number of possible fits decreases to 55. None of these fits is catastrophic:

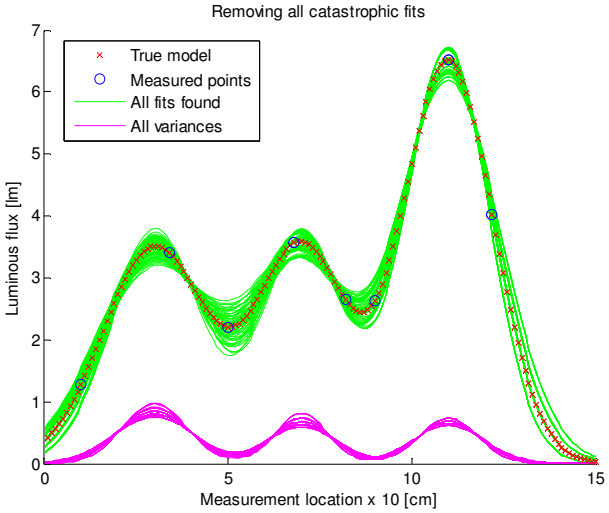


Figure 9.4 Reducing group of fits

The maximum of variances for this group of fits lies between 0.753 and 0.972. Therefore mean value of 0.85 was selected as the stopping criterion for this class of models.

10 PRACTICAL REALIZATION

10.1 The controlled robot

The application described in this thesis requires use of an autonomous robot which is able to move in space, run on battery, provide sensor measurements and communicate wirelessly with the base station (a PC in this case). For this purpose a device called iRobot Create (Roomba) has been chosen. The robot is built of plastic chassis with 3 wheels, mechanical and electronic parts and an electronic interface available to user. If user wishes, he can fully control the robot via this interface and also make use of the built-in microcontroller with an AD converter and analog and digital I/O [21]. The power to robot is supplied from an exchangeable rechargeable accumulator.

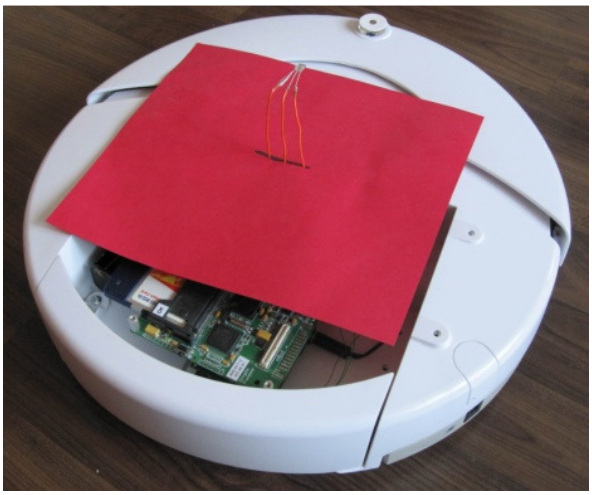


Figure 10.1 The controlled robot (iRobot Create or Roomba)

Besides very affordable price another advantage is the accessory called Home Base. When the battery power decreases robot's built-in logic can automatically navigate it and dock in the Home Base and initiate the recharge cycle.

Figure 10.2 Home Base – robot's recharging station

10.2 Wireless communication

As the main computational part of the application is running on a PC, a wireless communication link needs to be established with the robot. For this task we are using a Stargate single board embedded computer running Linux with a Compact Flash format Wi-Fi card. The embedded computer is placed in the robot's cargo bay. Connection between the robot and this computer runs via serial port, therefore a small application for the Stargate had to be written and compiled in Linux cross-compiler to translate between the TCP/IP link from the PC to the Stargate and serial port from Stargate to robot [11, 12].

Figure 10.3 StarGate Linux computer with Ambicom Wi-Fi card

10.3 Image recognition

A 640x480 wireless IP camera is placed above the scene. An algorithm for scene recognition has been programmed in C# and .NET platform.

Important objects – the borders and robot were labeled with a distinguishing color. Black dots in the Fig. 10.4 mark the robot and its space's corners after they were recognized and their coordinates saved. The algorithm compensates for the distortion caused by the misalignment of the camera relative to the center of the scene with the use of known distances between the corner points.

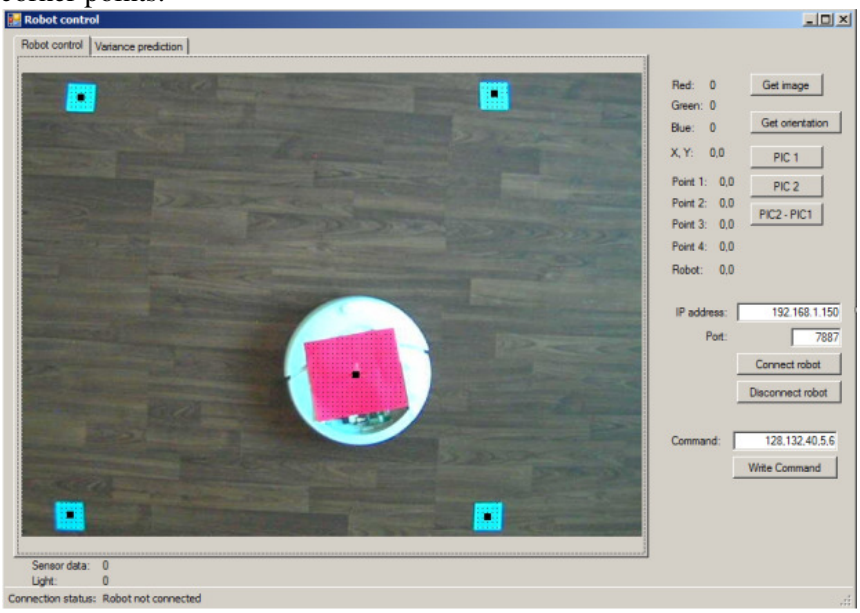


Figure 10.4 C#/.NET application for robot control, image recognition, model and variance calculation

A .NET (C#) application has been created in order to simplify the process of finding the model with smallest least-squares error, calculate its variance and find a new measurement point to minimize the variance (see Fig. 10.5):

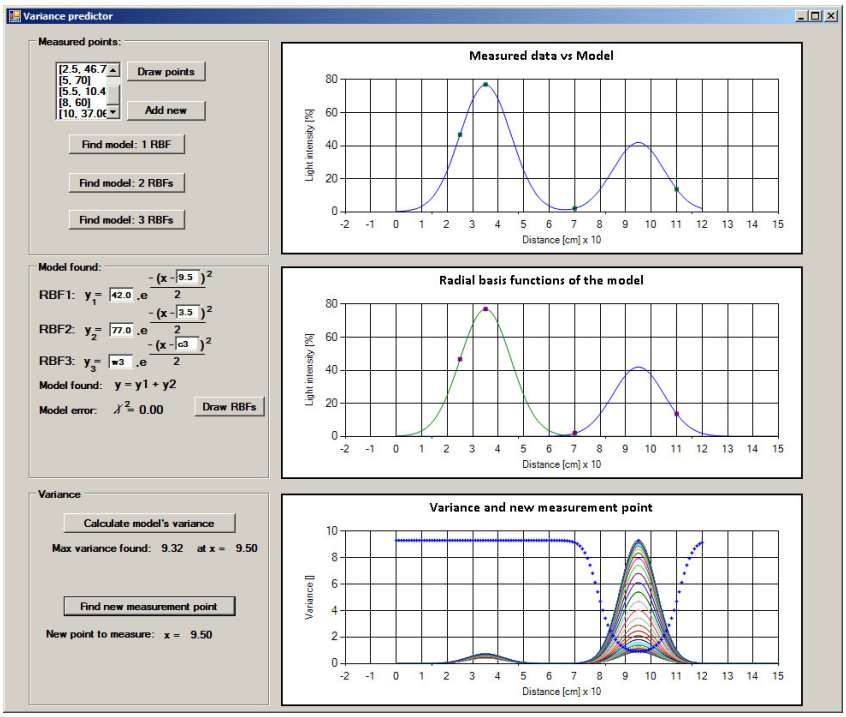


Figure 10.5 Model calculation and variance minimization

The navigation core with image processing described in Chapter 10.4 controls the robot to reach its destination, which is calculated by the variance-sensitive algorithm

10.4 Navigation and image processing

The purpose of image processing in this work is to obtain the current location of the robot and sensor placed atop of it. The number of methods are available for this task. The simplest approach is thresholding a black-and-white image. This works well under good light conditions and for the Roomba-a large white round object. The other simple alternative is to detect moving Roomba by differencing successive frames. The center of gravity is the estimate of current location. The Roomba can be navigated in four basic directions. At the start of navigation the Roomba orientation and the unit translation from the camera space into the Roomba space is unknown. The parameter equation for distance scaling and rotation can be written as

$$\begin{bmatrix} x \\ y \end{bmatrix} = A.k \begin{bmatrix} x \\ y \end{bmatrix} \tag{10.1}$$

$$A = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{10.2}$$

The A in the Fig. 10.6 is original location of the robot. Point B together with point A determines the vector ***u***, which is robot's current direction. Vector ***v*** given by points A and C is the orientation to which robot needs to be turned in order to reach the point C, which is the target location (next measurement point).

Figure 10.6 Calculation of angle to turn and distance to travel

In the (10.2) A is the orthogonal transformation matrix that represents a rotation by an angle α and k is the scaling between the image screen and physical space. In the simplest scenario the estimation of scaling k and rotation angle α can be separated. In general k is positional dependent on the location. The angle α represents the rotation movement of the robot and the difference between the physical and screen navigation directions. The least-square method leads to simple averaging of

$$\| AB \| = k \| AC \| \tag{10.3}$$

$$\alpha = arccos\left(\frac{\boldsymbol{u}.\boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|}\right) \tag{10.4}$$

and gives a good estimation of the two parameters [8].

A brief description of the navigation algorithm starts by recognizing coordinates of the robot's center (light-sensor location). However, this value itself is not sufficient to determine the robot's direction. This orientation is obtained by moving robot directly forward by 10cm and returning back to its original position. This movement determines the vector ***u***. From this vector and the vector ***v*** the angle α is calculated. Robot is then turned towards the point C and moved by the distance between point A and point C according to the law of Pythagoras.

The variance prediction algorithm running on the PC outputs the location of the next measurement in the *(x, y)* coordinates and sends it to Roomba.

As soon as the robot reaches its destination C another image is taken, current position recognized and the whole navigation procedure is repeated on the small scale now to minimize the error caused by the mechanics of the robot. This approach results in very

accurate navigation, although the work on the implementation has not been fully completed due to time limitations.

11 CONCLUSIONS

The presented doctoral thesis is aimed into area of system and model identification and utilizes statistical approaches and methods of artificial intelligence.

The common engineering modeling problem works with large or sufficient number of measurement points. In situations where the application-specific conditions do not allow arbitrary number of measurements to be taken, it is necessary to use an approach which guarantees a reliable model with minimal number of measurement points. Hence one of the goals of this thesis is development of such a method.

The first part of the thesis in Chapter 1 to Chapter 7 summarizes methodology used in modeling and system identification. The topics of model estimation and approximation with radial basis functions are examined first. The following chapters are focused on description of parameter estimation and least squares method. The background research part ends with variance inferences and introduction to genetic algorithms.

Based on the analysis and review of the materials presented in the first part Chapter 8 describes a situation of the static physical field whose parameters need to be identified and the model has to be constructed. In this chapter a method and algorithms using variance and predictive variance is presented. The linear parameters of the model are optimized using least-squares method and the nonlinear parameters are found with genetic algorithm. Also programmatic implementation is presented here. Chapter 9 shows a procedure of finding the maximum admissible value of variance that guarantees that the model found can be considered as non-catastrophic. Chapter 10 gives an overview of the tools used for practical verification of the results. The presented numerical examples in Chapters 8 and 9 show that the proposed methods and algorithms can be used in model identification and provide a way of finding a reliable model with small number of data points.

The theoretical and the novel contribution of this work consists of elaborating the method for use of variance in modeling and use of genetic algorithm for optimal parameter estimation.

The practical contribution in Chapter 10 lies in implementation of these algorithms in MATLAB and partially also in C#, together with the robot control and image recognition.

Further work on the topic of this thesis could focus on extension of the search for maximum admissible variance for different classes of models and basis functions, creation of their database and handling of situations where incorrect type of basis functions is chosen by user.

ZÁVĚR

Předložená doktorská dizertační práce je zaměřena do oblasti systémové identifikace a modelování a využívá statistické přístupy a metody umělé inteligence.

Častý inženýrský modelovací problém pracuje s dostatečným počtem měřicích bodů. V situacích, kde požadavky aplikace neumožňují provedení libovolného počtu měření je

potřeba použít přístup, který zaručuje spolehlivý model s minimálním počtem měřicích bodů. Vyvinutí takovéto metody je tedy jedním z cílů této práce.

První část dizertace v kapitolách 1 až 7 popisuje metodologii a terminologii využívanou v modelování a systémové identifikaci. Nejdříve jsou probrány oblasti sestavování modelu a aproximace pomocí radiálních bázových funkcí. Následující kapitoly se zaměřují na popis hledání parametrů a metodu nejmenších čtverců. Přehledová část je zakončena kapitolou odvozující varianci a úvodem do genetických algoritmů.

Na základě analýzy a zhodnocení materiálů uvedených v první části práce kapitola 8 popisuje situaci statického fyzického pole, jehož parametry je nutno identifikovat a vytvořit jeho model. V této kapitole je prezentována metoda a algoritmy využívající variance a prediktivní variance. Lineární parametry modelu jsou optimalizovány pomocí metody nejmenších čtverců a nelineární parametry jsou nalezeny genetickým algoritmem. Zároveň je zde uvedena část popisující programové řešení a implementaci. Kapitola 9 prezentuje proceduru hledání přípustného maxima variance, které zaručuje, že model lze považovat za spolehlivý. Kapitola 10 podává přehled vybavení a nástrojů použitých k praktickému ověření výsledků. Výsledky a příklady v kapitole 8 a 9 dokazují, že navržené metody a algoritmy mohou být použity při hledání modelu a poskytují způsob nalezení spolehlivého modelu s malým počtem měřicích bodů.

Původní teoretický přínos této práce spočívá ve vypracování metody využívající variance při hledání modelu a genetického algoritmu při optimalizaci jeho parametrů.

Praktický přínos popsaný v kapitolách 8 až 10 spočívá v návrhu programů v Matlabu a jazyku C# a také v řízení robota v prostoru na základě rozpoznávání obrazu získaného z kamery.

Další práce navazující na předmět této dizertace by se mohla zaměřit na rozšiřování vyhledávání přípustné hodnoty variance pro různé třídy modelů a bázových funkcí a tvorbu jejich databáze, a řešení situací, k nimž dochází v případě chybné volby počátečních podmínek expertním uživatelem.

REFERENCES

[1] Ljung, L.: *System Identification: Theory for the User*. P T R Prentice Hall, ISBN 0-13-881640-0, Eaglewood Cliffs, NJ, USA, 1987.

[2] Ljung, L.: *Perspectives on system identification*. 17th IFAC World Congress, ISBN 978-3-902661-00-5, Seoul, Korea, 2008.

[3] Bishop, C., M.: *Neural networks for pattern recognition*. Clarendon Press, ISBN 0-19-853864-2, Oxfor, United Kingdom, 1995.

[4] Buhmann, M., D.: *Radial Basis Functions: Theory and Implementations, Cambridge University Press,* ISBN 0521633389, Cambridge, United Kingdom, 2003.

[5] Lukaszyk, L.: *A new concept of probability metric and its application in approximation of scattered data sets*. Computational Mechanics Vol.33, Nr.4.

[6] Powell, M., J., D.: *Radial basis functions for multivariable interpolation: a review*. In J. C. Mason and M. G. Cox (Eds.), *Algorithms for Approximation,* pp. 143-167. Oxford: Clarendon Press, 1987.

[7] Michelli, C., A.: Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximations* 2, pp.11-22, 1986.

[8] Astrom, K.,J., Wittenmark, B.: *Adaptive Control*. Prentice Hall, ISBN 0201558661, 1994.

[9] Sorenson, H., W.: *Parameter estimation: Principles and problems*, M.Dekker, ISBN 0824769872, New York, USA, 1980.

[10] Anděl, J.: *Matematická statistika* (in Czech), SNTL/ALFA, Prague, Czech Republic, 1978.

[11] Bencúr, A., Šmíd, J., Kotzian, J., Pokorný, M.: *Measurement and modeling in sensor networks,* 9th RoEduNet International conference, ISBN 978-1-4244-7335-9, Sibiu, Romania, 2010, pp. 424-429.

[12] Bencúr, A., Šmíd, J., Pokorný, M., Kotzian, J.: *Variance and model optimization in sensor networks,* Applied Electronics, ISBN 978-80-7043-865-7, Pilsen, Czech Republic, 2010, pp.27-30.

[13] Smid, J.: *Informatics – modeling lectures,* ČVUT, Prague, Czech Republic, 2009.

[14] Anderson, T., W., Taylor, J.: *Strong Consistency of Least Squares Estimates In Dynamic Models*. The Annals of Statistics. 1979, Vol.7, No.3, 484-489.

[15] Smid, J., Volf, P.: *Variance Prediction for Sensor and Dialong Network Applications*, ICOMP'09/PSMP6, Las Vegas, USA, 2009.

[16] Mitchell, M..: *An introduction to genetic algorithms.* The MIT Press, ISBN 0−262−13316−4 (HB), Cambridge, Massachusets, USA, 1998.

[17] Lindfield, G., Penny, J.: Numerical Methods Using MATLAB. Ellis Horwood, USA, 1995.

[18] Smolensky, P.: *On the proper treatment of connectionism*, Behavioral and Brain sciences 11, nr.1: 1-14.

[19] Riolo, R. L.: *Survival of the fittest bits.* Scientific American*, 267,* no. 1: 114–116, 1992.

[20] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison−Wesley, 1989.

[21] Kurt, T., E.: *Hacking Roomba*, Wiley Publishing, Inc., ISBN 0-470-07271-7, Indianapolis, IN, USA, 2009.

LIST OF AUTHOR'S PUBLICATIONS

1. Bencúr, A., Kašík, V.: *Compact data acquisition module for the applications of meteorological measurements,* In IFAC PDS 2004, ISBN 83-908409-8-7, Krakow, Poland, 2004, pp. 299-303

2. Bencúr, A., Pokorný, M., Šmíd, J.: *Communication of autonomous agents based on wireless sensor motes*. In Proceedings of WSEAS 2005, ISBN 960-8457-12-2, Prague, Czech Republic, 2005, pp. 49-53**,** Indexed in ISI WOK

3. Bencúr, A., Pokorný, M., Šmíd, J.: *Embedded web servers and intelligent sensors in building-monitoring experiment*. In Proceedings of IWCIT 2005, ISBN 80-248-0906-0, Ostrava, Czech Republic, 2005, pp. 207-210

4. Bencúr, A.: *Autonomous sensors with embedded web servers*. In WOFEX 2005, ISBN 80-248-0866-8, Ostrava, Czech Republic, 2005, pp. 198-203

5. Šmíd, J., Obítko, M., Bencúr, A.: *Concept and sensor network approach to computing: The lexicon acquisition component*, 2nd International Workshop on Radical Agent Concepts, ISBN 960-8457-12-2, Greenbelt, Maryland, USA, 2005, Indexed in ISI WOK

6. Bencúr, A., Šmíd, J., Kotzian, J., Pokorný, M.: *Measurement and modeling in sensor networks,* 9th RoEduNet International conference, ISBN 978-1-4244-7335-9, Sibiu, Romania, 2010, pp. 424-429, Indexed in ISI WOK

7. Bencúr, A., Šmíd, J., Pokorný, M., Kotzian, J.: *Variance and model optimization in sensor networks,* Applied Electronics, ISBN 978-80-7043-865-7, Pilsen, Czech Republic, 2010, pp. 27-30, Indexed in ISI WOK

Reports of the solved research projects and participations

8. Bencúr, A. - Pokorný, M.: *Development of system for qualitative and semi-qualitative modeling,* author*,* Report of grant of FRVŠ, 2006, id. 2932/05/*G1*, VŠB – Technical University of Ostrava, 2006

9. GAČR 102/05/H525: *Rationalization of the doctoral study program at FEECS VŠB - TU Ostrava*, member of the PhD students' team, 2005

10. GAČR 102/03/1128: *Rationalization of meltage control in steel-mill with use of expert systems*, co-worker, VŠB – TU Ostrava, 2005

11. GAČR 102/06/1332: *Computational intelligence in control of metallurgical processes*, co-worker, VŠB – TU, Ostrava, 2006